제2권 통권7호 1989년 6월 1일 발행 (매월 1일 발행) 1988년 10월 10일 등록 (등록번호 라-3794) ㉾140-150 서울특별시 용산구 갈월동 17

# 하이테크 퍼스널컴퓨터

89/06

**특집/ IO.SYS의 분석**

디바이스 드라이버의 종류/설치/구조/호출/코맨드 패킷/
BPB/입출력 요구명령 처리루틴/램디스크

**유틸리티/** 터보에디터의 장점을 모두 갖춘 새로운 에디터 BBUN/
퍼스널컴퓨터를 이용한 취업정보 안내 프로그램/
자유롭게 텍스트 화면을 하드카피 해주는 유틸리티 레트릭스

**활용** 램디스크 플래그 레지스터를 읽기 쉽게

**테크닉/** 텍스트 화일을 프린트로 출력해 주는 유틸리티 TPRINT/
NKP에서의 윈도우 구현

**제작코너** 음성응답장치




정기구독자에게 프로그램디스켓을 무료로 보내드립니다.

□ 편집자 노트 □

# 모두가 눈을 똑바로 떠서 소비자를 우롱하는 기업을 몰아내야

벌써 제주도 전역에는 노오란 유채꽃이 만발하였습니다. 섬 전체의 1/4이 노란 물결로 뒤덮혀 마치 한폭의 아름다운 수채화를 보는 것처럼 싱그러움을 느끼게 합니다.
내 조국, 아름다운 우리나라를 사랑하는 마음은 누구나가 같은 마음일 것입니다.
얼마전부터 일기 시작한 컴퓨터 붐에 따라 우리 주위에도 여러가지 변화가 일어나고 있습니다.
컴퓨터의 대중화가 시작되면서 일기 시작한 각 기업의 가격인하바람은 앞으로도 당분간 계속될 것 같습니다.
소비자의 입장에서 보면 매우 바람직한 현상입니다. 그러나 아직까지도 일부 기업에서는 교묘한 방법으로 소비자를 우롱하고 있습니다. 생산 효율을 높혀 가격을 인하시켰다느니, 기술을 향상시켜 옛날과 같은 가격으로도 훨씬 성능을 높혔다느니 하며 과대선전을 하고 있으나 자세히 살펴보면 광고문안부터 소비자들이 제품을 제대로 파악할 수 없게끔 어지럽게 만들어 놓아 컴퓨터를 잘 모르는 일반인의 경우 제품성능을 제쳐놓고 기업의 신뢰도 내지는 광고문안 몇자에 비싼 비싼 고가품을 선택해야 하는 실정입니다.
그러나 그것도 잠시뿐 제품을 구입하여 막상 활용하려면 여러가지 옵션품목들을 부착하여야만 제대로 사용할 수 있습니다. 어떤 경우는 이들 주변장치들을 구입하는데 드는 비용이 오히려 훨씬 많이 드는 경우도 있습니다.
참으로 어이없는 경우입니다. 그러나 이러한 일들이 우리 주변에서 많이 일어나고 있습니다.
따라서 소비자들의 불신은 자꾸 쌓여가고 있는 실정입니다. 지금은 컴퓨터가 대중화되기 시작하는 초창기이므로 아직까지는 소비자들의 힘이 뭉쳐지지 않았을 때이지만 지금부터 그런 이미지를 심은 업체들은 머지 않아 소비자들로 부터 외면당하는 시기가 올 것입니다.
또 이러한 컴퓨터 열기를 틈타 많은 기업들이 컴퓨터쪽에 관심을 보이고 있습니다.
지금까지 전혀 컴퓨터쪽에 관계하지 않던 기업들도 컴퓨터란 글자만 들어가면 마치 큰 돈을 벌 수 있는 것처럼 착각을 하고 너도 나도 컴퓨터를 수입하여 판매하고 있습니다.
이러한 경쟁에는 대기업이나 중소기업이나 할 것없이 전부가 참여하고 있습니다.
비근한 예를 들어보면 애플 컴퓨터의 창시자 스티븐 잡스가 세운 NeXT사에서 발표한 신기종을 국내의 대기업을 비롯하여 10여개 업체에서 서로 수입하여 판매하려고 각축전을 벌이고 있습니다.
이 얼마나 부끄러운 일입니까?
돈을 버는 것도 좋고 소비자를 우롱하는 것도 좋다지만 지금 우리의 현실이 이제 겨우 컴퓨터에 눈을 뜨고 있는 실정임을 감안할 때 정말 부끄럽게 생각됩니다. 대기업이건 중소기업이건 모두 나서서 우리 손으로 만든 컴퓨터를, 우리 기술을 하루 빨리 익혀 소비자들에게 떳떳한 제품을 내놓아야 할 때인데도 불구하고 현실이 이러함을 생각하면 가슴이 답답합니다.
따라서 소비자들이 하루속히 눈을 떠서 철저히 이러한 — 기술 개발을 게을리 하고 소비자를 우롱하는 — 기업들을 몰아내야 할 것입니다.
그러한 시기가 와야 올바른 컴퓨터 풍토가 조성되고 정보화사회가 이루어질 것입니다.

# 유용한 유틸리티 2
# 사이드킥

구영미/ 서울대 컴퓨터공학과 대학원

바쁘게 변해가는 사회는 현대인의 빈약한 책상위에 너무도 많은 것을 요구한다. 그날 그날 처리해야할 문서나 보고서는 쉴새없이 밀어닥쳐 해결을 기다리고, 그러한 서류를 작성하는 중에 계산기나 메모리가 필요하여 이곳저곳 뒤적이며 찾기도 하고, 약속 시간에 맞추어 전화도 걸어야 하는데 깜박 잊는 등의 경우가 생길 수도 있다.

이렇게 한꺼번에 여러개의 작업을 관리해야 하는 상황에서 생각해 낸 것이 필요한 도구를 한 곳으로 모아 동시에 관리할 수 있는 방법이다. 보고서를 작성하면서 계산기를 사용하기 위해 서랍을 뒤지며 찾지 않아도 되고, 전화를 걸기 위해 자리를 뜰 필요가 없도록 한 프로그램이 볼랜드사가 1984~1985년에 걸쳐 상품화시켜 내보낸 사이드 킥이다.

이것이 국내에 소개된지는 오래되었으므로 여기에서 필자는 새로운 유틸리티의 보급이라는 측면보다는 기존 유틸리티의 간단한 사용법을 요약하며 초보자에게 사용상의 편리를 도모하고, 그 개발가능성을 검토해 가는 방향으로 설명하고자 한다.

## 특징

사이드 킥의 가장 획기적인 아이디어는 하나의 화면을 사용하여 여러개의 화면을 사용하고 있는 것처럼 쉽게 일할 수 있는 환경을 제공하는 윈도우 유틸리티의 개념이다. 그리하여 화일 편집을 하면서 계산기를 불러 계산을 하기도 하고, 캘린더로 잊지 않아야 할 사항을 확인하면서 필요하다면 전화도 걸 수 있는, 각 개인의 모든 사무를 관리하는 데스크 탑 오퍼레이터의 위치, 그것이 사이드킥의 기본역할이다.

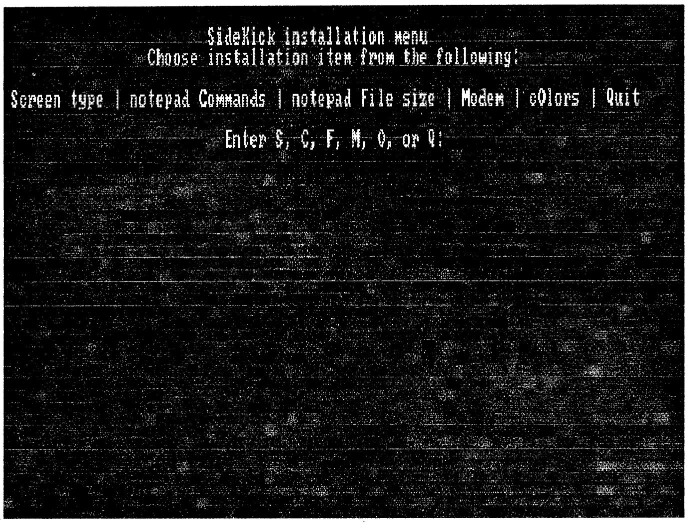
## 설치(Install)

현재 시스템을 구비하고 있는 대부분의 독자가 단색의 화면을 구비하고 있다고 생각해 볼 때 사이드 킥을 그냥 실행시켜도 무리는 없겠지만 통신이라든지 에디터를 사용하여 커다란 화일을 작성한다든지 혹은 컬러 모니터로 교체할 경우를 위해 효과적인 사용방법을 익혀두는 것이 유익하므로 설치부분을 먼저 설명하고 기능에 관한 부분은 조금후에 설명하겠다.

대체로 어떤 프로그램을 이용하려면 그 특성에 따라 자신의 시스템과 프로그램의 환경을 체크하는 것이 하나의 규칙이 되고 있다. 특히 사이드 킥과 같은 다양한 기능을 제공하기 위해 외부와 연결되어야 하는 유틸리티일수록 고려해야 하는 것이 설치작업이다. 사이드 킥을 수행시키기 전에 스크린형, 에디터명령어, 에디터의 화일크기, 모뎀포트, 컬러 등을 사용자의 시스템에 맞게 환경을 구비하여야 하는데 우선 설치하기 전에 유의할 것은 사용하고 있는 컴퓨터의 모니터, 화면 표시 방식, 통신 포트 등이 다음의 어디에 속하는가를 확인해야 한다.

① 모니터 어댑터 : 단색, 컬러, 단색과

〔화면 1〕 설치 주메뉴



컬러둘다.
② 내정된 화면표시 방식 : 단색 화면, 컬러 화면(80×25)
③ 모뎀에 이용되는 비동기식 통신 포트 : COM1, COM2

이제 SK.COM, SKN.COM, SKM.COM 등의 프로그램이 사용 드라이브 내에 있는가를 확인한 후 다음과 같이

```
SKINST ↵
```

라고 입력하면 화면 1과 같은 설치 메뉴가 나타난다. 설치작업은 화면 1의 주메뉴에 나타난 S, C, F, M, O, Q 중에 필요한 항목을 선택하여 한다.

### (1) 스크린 형(S)

설치 메뉴에서 S를 누르면 사이드 킥의 스크린 방식을 묻는 메시지가 나온다. 0~3중에서 선택하여 해당 번호를 입력하면 되는데 0번은 내정된 화면표시, 1은 단색 화면표시, 2는 컬러 화면표시, 3은 B/W 화면표시를 나타낸다.

### (2) 에디터 명령어(C)

사이드 킥의 에디터인 Notepad 상에서 커서를 움직이고 텍스트를 관리하는 작업은 한개나 그 이상의 키를 조합하여 하고 있는데 이 키들은 터보 파스칼과 워드스타 등에서 표준으로 정의되어 바꿀 수 없는 부차적명령어(Secondary command)와, 키보드상의 펑션 키와 화살표키 등을 이용하여 만든 주명령어(Primary command)로 나누어진다. 설치를 통하여 변경할 수 있는 것은 후자의 주명령어이다.

현재의 커서위치를 바꾸고자 할 때 설치 주 메뉴에서 C를 누르면 다음과 같은 메시지가 나온다.

```
CURSOR MOVEMENTS :
1:character left <ESC> K -)
```

여기서 커서를 왼쪽으로 옮기는 키를 〈ESC〉 K에서 〈ESC〉 C로 바꾼다

〔표 1〕 사이드킥의 수행 화일

| 이 름 | 제공하는 기능 |
|---|---|
| SK. COM | 모든 기능을 갖춘 완전한 Sidekick시스템 |
| SKN. COM | Notepad, Calculator, Dialer, ASCII |
| SKC. COM | Calendar, Calculator, Dialer, ASCII |
| SKM. COM | Calculator, ASCII |

면→다음에 수정하고자 하는 값을 입력하면 된다. 이때 입력방법은 사용하고자 하는 키를 직접 입력하는 방법과 명령어를 이루는 문자의 ASCII 값을 입력하는 방법 중에서 임의로 택하여 입력한다. 입력후에는 ↵ 를 눌러 종결시킨다.

### (3) 에디터 화일크기(F)

에디터를 사용하여 화일을 만들려면 크기가 작을 경우는 상관없으나 화일이 커질 경우는 그 크기를 지정해야 한다. 사이드 킥의 Notepad는 그 텍스트를 전적으로 주메모리 내에 두기 때문에 Notepad 화일에 필요한 공간이 메모리내에 마련되어야 한다. 내정된 화일크기는 4000문자이고 허용범위는 1000~50000 문자이므로 메모리 문제와 화일크기를 고려하여 잘 선택해야 한다.

### (4) 모뎀(M)

Dialer를 사용할 때 사이드 킥에게 모뎀이 COM1과 COM2 중 어느 포트위에 설치될 것인가를 알려주기 위해 사용된다.

### (5) 컬러(O)

컬러의 설치는 컬러방식과 B/W방식, 단색방식 등에서 윈도우 프레임과 텍스트 등에 사용되는 색깔을 바꾸는 일을 한다.

화면 중앙에 색깔의 값을 해당 색깔로 보여주므로 이것을 참고로 하여 윈도우 하단에 있는 항목을 선택하여 값을 바꾸면 된다.

## 로드와 제거

설치 작업을 완료시킨 후에는 실행시켜 보자. 사이드 킥은 램상주 프로그램으로서 일단 메모리에 로드되면 컴퓨터를 끌 때까지 주 메모리에 머물러 있으므로, 필요할 때는 언제든지 호출하여 사용할 수 있다. 사이드 킥을 메모리에 로드시키기 위한 실행화일과 그 기능은 표 1에 소개된 것들인데, 이 중에서 하나를 선택하여 실행시키면 된다. 예를 들어 모든 기능을 갖춘 완전한 사이드 킥 시스템을 램에 로드시키고자 한다면 SK.COM이 있는 상태에서

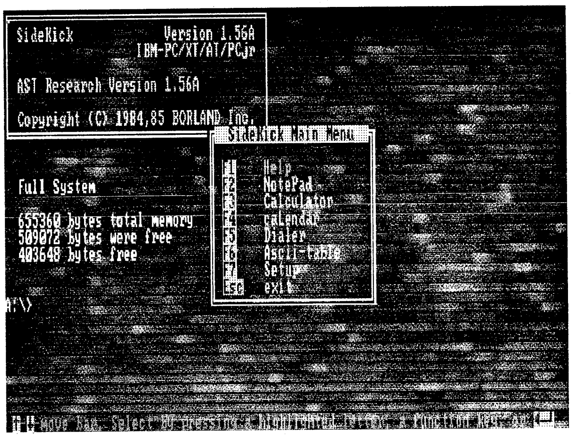
```
SK ↵
```

라고 입력하면 된다. 이때 화면 전체가 지워지고 화면 좌측상단에 로그 온 메시지가 나타날 것이다. 로그 온 메시지의 내용은 시스템이름, 버전번호, 사이드 킥 시스템의 설명과 메모리에 관한 정보를 나타낸다.

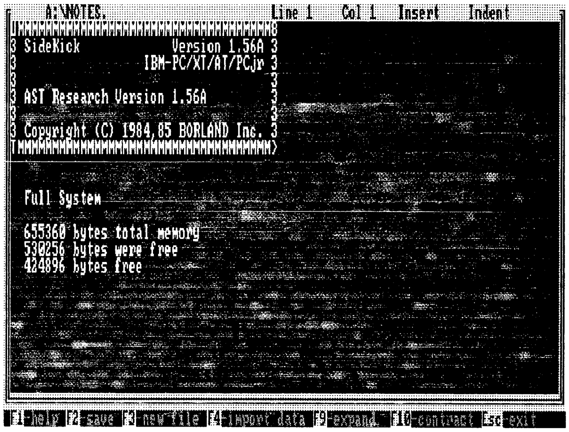
로그 온 메시지가 나타나면 램 메모리내에 SK.COM이 로드된 것이므로 Ctrl키와 Alt키를 동시에 누르거나 키보드상의 두개의 시프트 키를 눌러서 화면 2와 같은 주메뉴 선택 윈도우를 불러 온다. 그리고 램에 있는 사이드 킥을 제거하고자 할 경우에는 Ctrl-Alt를 눌러 화면에 주메뉴가 나타나게 한 후 Crtl-Home-End를 동시에 누른다.

사이드 킥의 메뉴는 Help, Notepad, 계산기, 캘린더, ASCII, 셋업, Exit 등이 있는데 원하는 항목의 선택은 각 메뉴를 선택하는 문자의 첫글자를 입력하거나 해당 펑션키나 화살표키를 사

〔화면 2〕 주메뉴 선택 윈도우




〔화면 3〕 노트패드




용하여 수평바를 원하는 항으로 옮겨 리턴하면 된다. 특징적인 기능을 살펴보면 다음과 같다.

# Notepad

Notepad는 앞서 설명한 대로 사이드 킥의 에디터 역할을 담당하는 것으로 이것을 이용하여 문안을 작성하면 되는 것이다. 또한 이것은 터보 파스칼 에디터, 워드스타와 호환성을 가지므로 만약 이들의 에디터를 사용해본 경험이 있는 독자라면 쉽게 사용할 수 있을 것이다.

표 2에 명령키와 기능에 관하여 간단히 나타내었다. 에디터를 이용하여 화일을 작성할 때 표 2의 기능들을 사용해 보기 바란다. 여기에 추가된 기능에 대하여는 '⁎'와 '#'로 구분하였는데 '⁎'는 워드스타에 구현되지 않은 기능이고, '#'는 터보 파스칼에 추가된 기능임을 참고하기 바란다.

Notepad를 열게 되면 **화면 3**과 같이 하단에 펑션키들이 나타난다. 그 펑션키와 기능을 알아보면 아래와 같다.

F1 : Help

F2 : 저장

F3 : 새로운 화일 지정

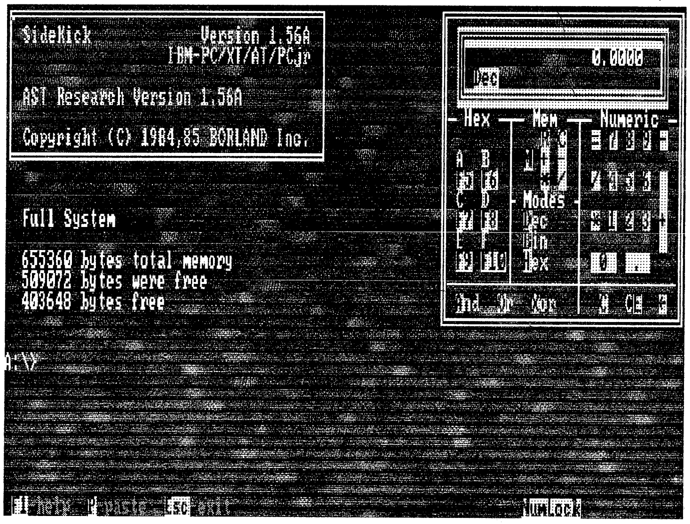
F4 : Notepad를 열기 이전 상태의 화면에서 필요한 블럭을 복사한다. Notepad를 사용하다가 이 키를 누르면 화면이 바뀌어 편집 이전의 화면이 나타나는데 이때 Ctrl-K-B와 Ctrl-K-K로 필요한 부분을 블럭마크한다. Ctrl-K-K를 누름으로 화면은 Notepad 상태로 환원되는데 여기서 커서를 블럭이 시작될 위치로 옮겨 블럭복사명령(Ctrl-K-C)을 누른다.

F9 : 윈도우 확장

F10 : 윈도우 축소

Notepad의 특별한 기능으로 Cut_and_paste가 있는데, 이것은 블럭을 하나의 키에 할당하여 블럭을 복사할 수 있게 하는 명령으로 블럭을 마크한 후 Ctrl-K-E를 누르면, 할당될 키에 대한 질문이 나온다. 이때 Alt-3과 같이 다른 용도로 쓰이지 않는 키를 선택하여 할당시킨다. 그 다음에 나오는 블럭페이스트, 라인페이스트에 관한 질문에 대해서는 B나 L을 눌러 답한다.

〔화면 4〕 계산기



〔표 2〕 편집 명령어와 키

| 동　　　　작 | 부차적 명령어 | 주 명령어 | PC키 |
|---|---|---|---|
| 커서 이동 | | | |
| 한문자 좌향 이동 | Ctrl-S | <Esc>K | ← |
| 한 문자 우향이동 | Ctrl-D | <Esc>M | → |
| 한 단어 좌향 이동 | Ctrl-A | <Esc>S | Ctrl-← |
| 한 단어 우향 이동 | Ctrl-F | <Esc>T | Ctrl-→ |
| 위로 한 줄 이동 | Ctrl-E | <Esc>H | ↑ |
| 아래로 한 줄 이동 | Ctrl-X | <Esc>P | ↓ |
| 위로 스크롤 | Ctrl-W | | |
| 아래로 스크롤 | Ctrl-Z | | |
| 위로 한 페이지 이동 | Ctrl-R | <Esc>I | PgUp |
| 아래로 한 페이지 이동 | Ctrl-C | <Esc>Q | PgDn |
| 라인의 제일 왼쪽으로 이동 | Ctrl-Q-S | <Esc>G | Home |
| 라인의 제일 오른쪽으로 이동 | Ctrl-Q-D | <Esc>O | End |
| 페이지의 꼭대기로 이동 | Ctrl-Q-E | <Esc>W | Ctrl-Home |
| 페이지의 바닥으로 이동 | Ctrl-Q-X | <Esc>U | Ctrl-End |
| 화일의 처음으로 이동 | Ctrl-Q-R | <Esc>Ctrl-D | Ctrl-PgUp |
| 화일의 끝으로 이동 | Ctrl-Q-C | <Esc>V | Ctrl-PgDn |
| *#화일 끝으로 이동하여 날짜/시간 기록 | Ctrl-Q-O | | |
| 블럭의 시작점으로 이동 | Ctrl-Q-B | | |
| 블럭의 끝점으로 이동 | Ctrl-Q-K | | |
| 이전의 커서 위치로 이동 | Ctrl-Q-P | | |
| 삽입과 삭제 | | | |
| 삽입방식 on/off | Ctrl-V | <Esc>R | Ins |
| 라인 삽입 | Ctrl-N | | |
| 라인 삭제 | Ctrl-Y | | |
| 커서에서 라인의 끝까지 삭제 | Ctrl-Q-Y | | |
| 커서 오른쪽의 단어 삭제 | Ctrl-T | | |
| 커서 아래의 문자 삭제 | Ctrl-G | <Esc>S | Del |
| *왼쪽 문자 삭제 | | Ctrl-H | ← |
| 블럭 명령어 | | | |
| 블럭의 시작점 표시 | Ctrl-K-B | <Esc>A | F7 |
| 블럭의 끝점 표시 | Ctrl-K-K | <Esc>B | |
| *한 단어의 블럭 표시 | Ctrl-K-T | | |
| *블럭을 숨기거나 나타냄 | Ctrl-K-H | | |
| 블럭 복사 | Ctrl-K-C | | |
| 블럭 이동 | Ctrl-K-V | | |
| 블럭 삭제 | Ctrl-K-Y | | |
| 디스크로부터의 화일 입력 | Ctrl-K-R | | |
| 블럭을 디스크에 기록 | Ctrl-K-W | | |
| *#블럭 소트 | Ctrl-K-S | | |
| *#블럭 인쇄 | Ctrl-K-P | | |
| *#블럭 페이스트 | Ctrl-K-E | | |
| 기타 | | | |
| 노트 화일 저장 | Ctrl-K-D | | F2 |
| 탭 | Ctrl-I | | |
| 최근의 탐색 반복 | Ctrl-L | | |
| 제어문자의 접두사 | Ctrl-P | | |
| 탐색 | Ctrl-Q-F | | |
| 탐색과 치환 | Ctrl-Q-A | | |
| *자동 인덴테이션의 ON/OFF | Ctrl-Q-I | | |
| *라인 복구 | Ctrl-Q-L | | |
| 시간/날짜 기록 | Ctrl-Q-T | | |
| *#그래픽의 ON/OFF | Ctrl-Q-G | | |
| #오른쪽 한계 설정 | Ctrl-O-R | | |
| #단락의 재정비 | Ctrl-B | | |

Notepad 상단의 상태라인에는 화일 이름, 커서위치, 삽입방식, 자동 인덴테이션상태, 그래프의 ON/OFF 등의 정보가 기술되어 편집작업을 돕는다.

## 계산기

계산기는 **화면 4**에 나타난 것과 같이 사무용 계산기와 유사한 형태를 가지며 주선택 윈도우에서 F3나 C 등을 눌렀을 때 선택되어 진다. 계산기의 입력은 키보드 상단의 숫자필드를 이용하여 하며, 계산기 작동시에는 Num lock 상태가 자동적으로 ON 되어 있으므로 키보드 오른쪽의 뉴메릭 키패드를 사용할 수 있다. 이때 가감승제 4진 연산을 수행할 수 있다.

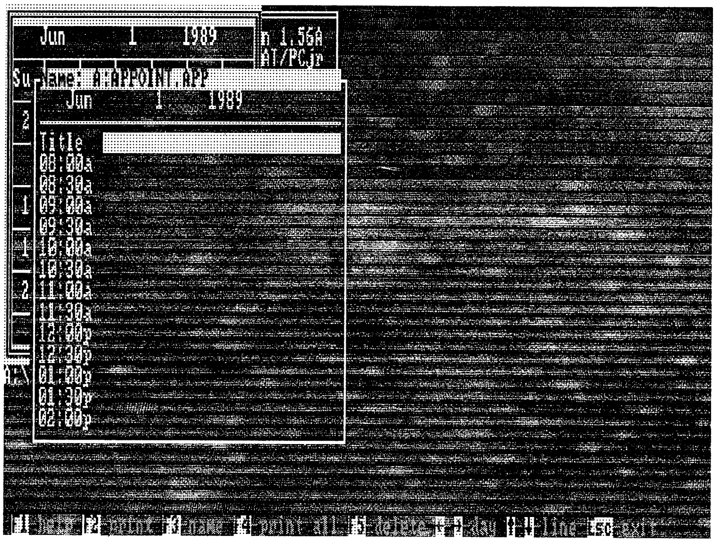
또하나 추가된 기능으로 계산된 결과의 숫자들을 페이스트 시킬 수 있는데 이러한 작업은 계산을 수행한 후에 P를 누르고 숫자를 지정해 주고 싶은 임의의 키를 눌러주면 이 값을 Notepad에서도 이용할 수 있다.

이 계산기 기능은 사무용 계산기와 문안 편집기를 한곳에 집중시켜 사무의 단순화 및 자동화를 도모했다는 면에서 볼 때 상당한 의의를 지닐 수 있으나 사인, 코사인, 지수, 로그조차 제공못하는 4진연산만의 계산기가 과연 어느 정도의 효용도를 지닐 수 있는 가에 관한 문제를 고려해 볼 때 상당한 부분의 보완이 필요하다.

## 캘린더

캘린더는 매일의 약속이나 계획을 기입하고 관리하는데 사용된다. 주선택 윈도우에서 캘린더를 선택하면 화면 좌측상단에 달력과 유사한 윈도우가 나타난다. **화면 5**에 나타난 캘린더 상단의 글자는 약속된 날짜를 나타내는데 만약 약속을 기입하고 관리할 날짜가 1989년 6월 1일이라면 이 상태에서 리턴하면 그 날짜에 해당하는 날의 약속란(Appointment pad)을 보고 그 내용을 삽입하거나 갱신할 수 있도록 한다.

〔화면 5〕 약속란을 불렀을 때의 캘린더




날짜의 변경은 캘린더의 하단에 나타난 ←와 →로는 달을 바꾸고, ↑나 ↓로는 해를 바꾸어 숫자를 입력하여 변경할 수 있다.

원하는 날짜로 변경되어 윈도우에 표시된 날짜가 더 이상의 수정이 필요 없을 때는 리턴하여 해당 날짜의 약속란을 갖는 새로운 윈도우를 부른다. 맨 윗줄은 그날의 타이틀을 기록하기 위한 줄이고, 그 다음 줄들은 오전 8시부터 30분 간격으로 약속을 기입할 수 있도록 되어 있다. 여기서 ↑와 ↓는 시간대를 선택하는데 사용되고 PgUp과 PgDn키는 각 날의 제 1페이지와 제 2페이지 선택에 쓰이고 ←와 →는 그 전날과 그 다음날을 선택하는데 쓰인다. 화면 하단에 나타나는 펑션키와 그 기능을 알아 보면 다음과 같다.

F1 : Help

F2 : 약속캘린더의 이름을 기입하여 새로운 이름의 약속캘린더를 불러오도록 한다.

F3 : 현재의 약속 캘린더의 일정기간 약속을 인쇄하는데 사용된다.

F4 : 현재의 약속 캘린더의 모든 약속을 인쇄하는데 사용된다.

F5 : 현재의 약속 캘린더의 일정기간 약속을 지우는데 사용된다.

## Dialer

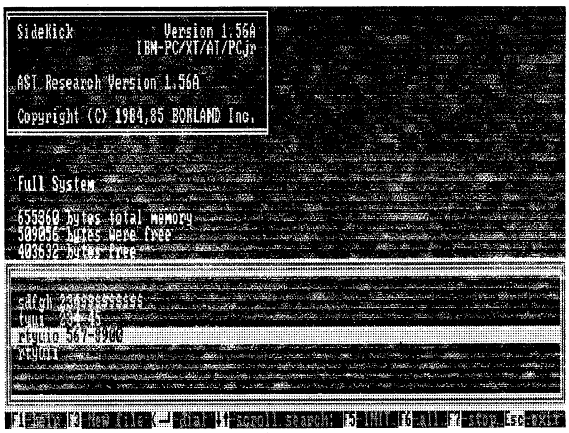
Dialer의 작업은 현재의 화면 또는 이미 관리하고 있는 전화 디렉토리 화일에서 전화번호를 찾아 번호를 걸어주는 것으로 Dialer의 작업을 보조해 주는 전화 디렉토리 화일의 각 라인의 형식은 다음과 같다.

INITIALS PHONENUMBER COMMENT

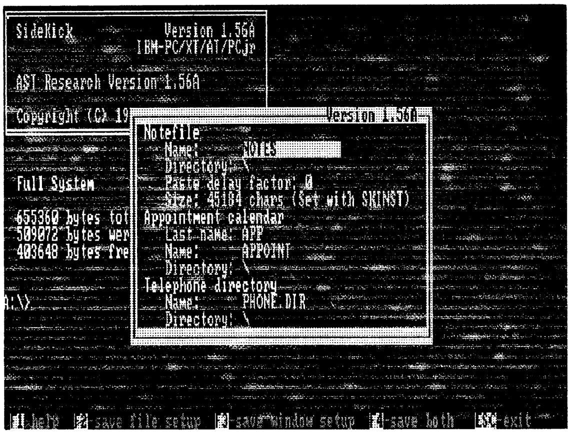
INITIALS는 사람의 이름이나 약어이고 선행 공백도 의미를 지닌 것으로 간주한다. PHONENUMBER는 특수문자(하이픈이나 괄호) 중의 하나를 포함해야 한다. COMMENT는 선택적인데 이것은 사람의 정식이름이나 주소일 수도 있고 각 사람과 관련된 정보일 수도 있다. 전화 디렉토리 화일은 Notepad와 같은 에디터로 하면 되며, 각 레코드 필드들은 쉼표나 공백으로 구분된다.

이렇게 작성된 화일로 전화를 거는데 사용하려면 주선택 메뉴에서 Dialer가 호출한다. 이때 Dialer는 먼저 화면 6과 같이 화면상에서 유효한 전화번호

〔화면 6〕 Dialer




〔화면 7〕 Set up




를 찾는다. 상단에서부터 찾아가 사용자가 걸고자 하는 전화번호를 판단하여 맞으면 리턴하고 아니면 →를 눌러 다음으로 넘어가도록 한다.

화면상의 전화번호로 전화하지 않고 Dialer 자체의 전화 디렉토리 화일을 이용할 때는 사용자가 스페이스바를 누른다. 또 화면에서 유효한 전화번호를 찾을 수 없을 때는 사용자가 스페이스 바를 누른 것처럼 자동적으로 윈도우가 나타나며 그 안에 전화 디렉토리화일이 보인다. 그러면 화살표 키를 가지고 화일을 아래 위로 훑어가다 원하는 전화번호가 나타났을 때에 리턴하여 전화할 수 있다.

Dialer 윈도우 하단의 펑션키의 기능은 다음과 같다.

F1 : Help
F2 : 새로운 전화디렉토리 지정
F3 : 머리글자로 탐색(선형 공백도 의미가 있으므로 입력한다)
F4 : 전화 디렉토리화일의 위치에 관계없이 화일 전체에 걸쳐서, 입력하는 단어를 찾아 그 단어가 있는 줄을 윈도우의 수평바가 가리키도록 한다.
F5 : 탐색정지

## ASCII

이 기능은 ASCII 값을 찾고자 할 때 쓰이며, 256개의 ASCII 문자세트 중에서 한번에 16개씩 또는 32개씩의 문자세트에 대해 10진값, 16진값, 화면 표시형태 등을 보여준다.

## 셋업

이 기능은 정해진 표준치를 바꾸는데 사용하는데(예를 들어, Notepad를 호출했을 때에 Notepad 윈도우에 제일 먼저 올라오는 화일은 NOTES라고 정해져 있는데 이를 다른 화일로 바꿀 수 있다), 바꿀 수 있는 항목은 **화면 7**에 보이는 것과 같다. 항목선택은 ↑나 ↓로 바를 옮겨 하는데 항목이 선택되면 변경할 사항을 입력한 후 리턴하여 변경을 고정시킨다.

항목중에서 화일이름에 해당되는 내용은 해당윈도우가 호출되었을 때, 자동적으로 로드되는 화일이고 디렉토리는 패스가 선언되지 않았을 때의 내정된 경로이다. 화면 하단에 나타나는 펑션키와 기능은 아래와 같다.

F1 : Help
F2 : Setup 윈도우에 나타나 있는 이름, 디렉토리, 지연요소 등이 사이드 킥에 저장
F3 : 윈도우들의 현재 위치와 Notepad의 현재의 위치가 사이드 킥에 저장.
F4 : 화일 셋업과 윈도우 셋업이 저장.

## 맺음말

이상에서 설명한대로 사이드 킥은 모든 정보와 도구의 집중화를 추구하려는 현대인의 심리에 착안하여 만들어진 소프트웨어로, PC에서 윈도우 소프트웨어의 영역확보라는 면과 변화하는 사회심리의 도입이라는 면에서는 그 실용성과 의의를 충분히 인정받을 수 있으리라 본다.

사이드 킥의 처음버전 1.11C에 비해 1.56은 Netepad상에서 F3를 눌러 새로운 화일를 로드시킬 때 이전에 기록했던 화일이 남아 있는 등의 약간 개선된 점이 있다. 그리고 더욱 개선된 형태로 사이드 킥 +가 나왔으나 이것은 사이드 킥의 장점중의 하나인 단순성을 배제시켰을 뿐 메뉴에서의 미비한 짜임새나 에디터에 비해 기타의 도구들의 효용면에서 Notepad의 보조역할 정도에 그치므로 별 개선점이 없다고 할 수 있다. 유틸리티의 개선은 우선 효율적인 이용에서 비롯된다는 것을 필자의 경험에서 얻어낸 결론인바 독자들의 날카로운 사용이 더욱더 편리하고 개선된 유틸리티가 개발될 수 있는 밑거름으로 작용하기를 기대한다. PC

# 컴퓨터 기본상식 [1]
# 비디오 어댑터

서장석 / 광운대학교 컴퓨터공학과 4년

우리가 컴퓨터를 쓰기 위하여 항상 제일 먼저 접하게 되는 비디오 어댑터의 원리를 알아보고 비디오 어댑터를 직접 콘트롤 할 수 있는 방법을 살펴보기로 하자. 상식적인 이야기이므로 부담을 갖지 말고 읽어주기 바란다.

## 비디오 어댑터란?

비디오 어댑터란, 이미지를 사람이 이해할 수 있도록 모니터화면으로 표현해 주는 장치를 말한다. 이 비디오어댑터의 종류는 매우 많으나 여기에서는 우리 주변에서 가장 많이 사용하는 한가지를 선택하여 알아보기로 하자. 필자의 경우도 예외는 아닌데 아마 대부분의 IBM/PC 호환기를 사용하는 사람은 Hecules 사에서 제작한 허클리스 그래픽 카드(HGC)를 사용하리라 생각한다.

이 HGC는 IBM 사의 모노크롬 디스플레이 어댑터(MDA)를 겨냥하여 내놓은 제품이다. 따라서 MDA보다 나은 성능을 가진 것은 당연한 일일 것이다. 세운상가에서 구할 수 있는 한글카드나 삼보의 한글카드 등 대부분의 한글카드도 이 HGC에 한글을 하드웨어로 처리해 주는 모듈을 올려 제작한 것이라서 HGC용 소프트웨어를 그대로 실행시켜 준다. 그렇다면 먼저 비디오 어댑터의 원리를 알아보도록 하자.

비디오 어댑터에는 크게 두가지 방식이 있다. 하나는 래스터 스캔방식이고 또 하나는 벡터 스캔방식이다. 이 두방식의 차이를 살펴 보려면 먼저 모니터의 원리를 알아야 한다. 그러나 우리는 비디오 어댑터에 대한 이야기를 주제로 삼고 있으므로 간단하게만 언급을 하도록 하겠다.

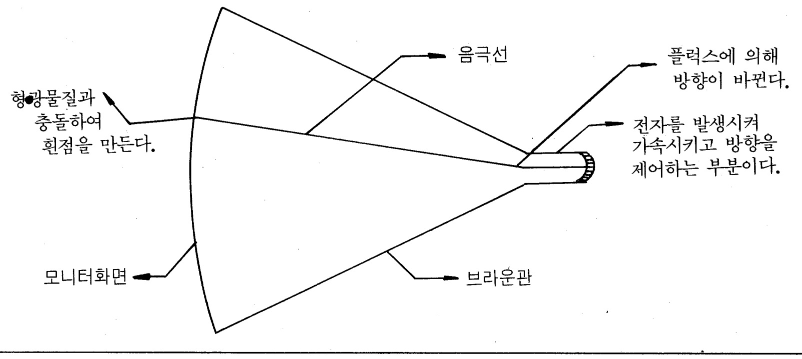
모니터에 화상이 생성되는 것은 전자총에서 출발한 전자가 가속을 받아 형광물질이 코팅되어져 있는 모니터의 화면에 부딪쳐 빛을 발함으로써 일어난다( 그림 1 참조). 전자는 전자총에서 끊임없이 만들어지므로 연속된 통로를 이루게 된다. 이 전자의 이동경로를 음극선이라 하기 때문에 브라운관을 다른말로 음극선 관이라 부르는 것이다.

이미지는 이 음극선의 방향을 플럭스로 제어해 줌으로써 형성된다. 예를 들어 이 음극선이 화면 한 부분에 계속적으로 비춘다면 우리는 스크린상에서 하나의 점만을 보게 될 것이다. 우리가 흔히 T.V를 껐을때 가운데 점을 보게 되는 것은 이 음극선을 제어하는 플럭스를 만들어내는 부분에 동력이 먼저 끊어지고 음극선은 그 후 약간의 시간동안 더 길게 방출되기 때문에 가운데만을 비추게 되는 것이다.

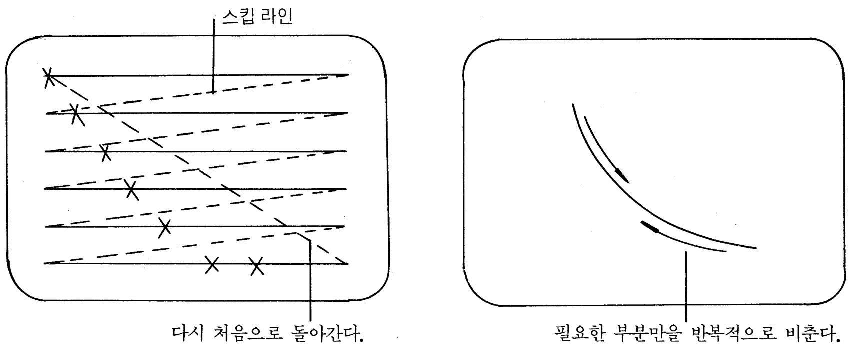
이처럼 모니터는 음극선을 이용해 화상을 형성한다. 음극선은 계속적으로 만들어지는 전자의 이동경로라 하였으므로 마치 소방차의 호스에서 뿜어지는 물줄기와도 같다. 이 물줄기는 소방관의 손에 의해 제어된다. 소방관이 호스의 각을 조금씩 바꾸어 줌으로써 물줄기는 다른 곳을 향하게 되는 것처럼 모니터 또한 전자가 발생되어 가속되어지는 부분에 제어를 위한 장치를 하고 이것을 이용해 음극선의 방향을 제어한다. 따라서 음극선의 표적은 화면 전체가 된다. 음극선이 출발하는 곳이 하나의 점이기 때문에 브라운관은 이 점을 중심으로 한 구형일 수 밖에 없다. 여하튼 모니터는 음극선을 사용해 화상을 형성한다는 사실을 알게 되었다.

그렇다면 래스터 스캔방식과 벡터 스캔방식의 차이를 알아보자. 먼저 래

〔그림 1〕 브라운관의 원리

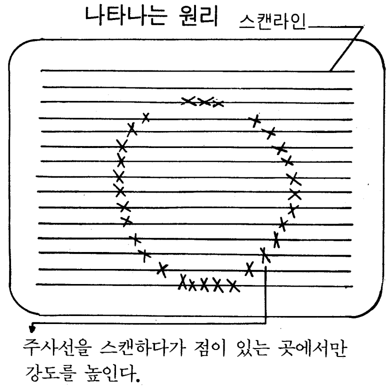


〔그림 2〕 두가지의 스캔방식




(a) 래스터 스캔방식 (b) 벡터 스캔방식

〔그림 3〕 래스터 스캔방식에서 도형이 나타나는 원리

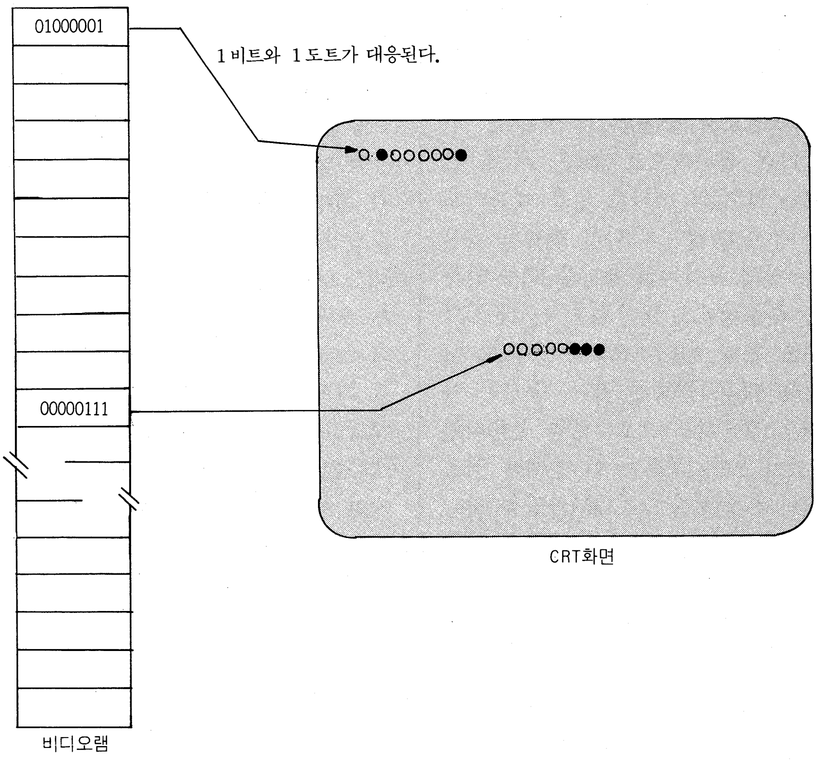



스터 스캔방식은 그림 2(a)에서 보인바와 같이 음극을 일정한 주기로 화면의 좌측 위에서 우측 아래로 스캔하고 필요한 부분만을 밝게 하는 방식이다. 이 음극선이 시작점(좌상)에서 끝점까지를 모두 스캔하는 시간은 대략 1/30초이기 때문에 모니터는 1초에 30회정도의 화상을 화면에 형성한다. 그러나 사람의 눈은 잔상이라는 효과가 있으므로 이 부분을 알아 차리지 못한다. 허나 모니터 화면을 카메라에 담아 다시 보면 그 화상 재생에 의해 생기는 화면 떨림을 발견할 수 있다.

그 다음으로 벡터 스캔방식은 래스터 스캔방식이 모든 점을 스캔하고 필요한 지점에만 밝은 빛을 발하는데 비해 필요한 부분에만 음극선을 비춘다. 따라서 좀더 선명한 화질을 얻을 수가 있다. 이 벡터 스캔방식에는 벡터좌표를 사용해야만 하는데 그 이유는 이 음극선의 이동경로를 벡터값으로 콘트롤 하기 때문이다. 만일 화면에 원이 하나 그려져 있다면 래스터 스캔방식의 모니터는 전체를 스캔하다가 그림 3과 같이 원이 있는 부분만을 밝게 비출 것이고 벡터 스캔방식은 원의 주위만을 돌면서 비출 것이다. 앞서 이야기한 HGC는 바로 래스터 스캔방식을 사용한 어댑터이다.

래스터 스캔방식의 어댑터는 앞서 설명한 것처럼 화면에 점이 있는 부분만을 밝게 비추는 일을 하기 때문에

〔그림 4〕 비디오램과 래스터 스캔화면




화면의 모든 점과 1대 1로 대응되어지는 비디오 램을 가질 수 밖에 없다(그림 4 참조). 이 비디오 램을 차례로 액세스하여 대응되는 비트가 1일때에 음극선의 강도를 높이게 된다. 그렇기 때문에 이 비디오 램의 데이터가 모두 0이라면 화면에는 아무것도 나타나지 않을 것이며 모든 데이터가 1이라면 화면은 백색이 될 것이다. 비디오램의 주사선 이동경로는 그림 5와 같다.

여기서 일단 정리를 좀 해 보면, 우리는 HGC의 원리를 알아보기 위하여 두가지 스캔방식을 이야기하였고 HGC는 래스터 스캔방식을 사용한다는 말도 하였다. 래스터 스캔방식이란 특징 때문에 비디오 램을 가져야 한다는 말을 하였고 따라서 모든 래스터 스캔방식의 어댑터는 이 비디오 램의 데이터

를 바꾸어 줌으로써 제어할 수 있다는 결론을 얻게 되었다.

## 비디오 램

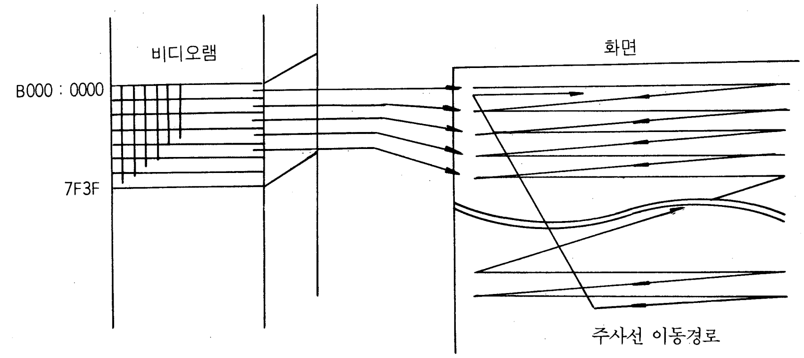
그렇다면 이 HGC의 비디오 램은 어디에 존재하는가? 여기에 대한 답은 이렇다. 우리가 흔히 메인 메모리라 함은 두가지로 해석을 한다. 우선 그 프로세서가 각가지의 어드레싱 모드를 이용하여 액세스할 수 있는 메모리를 메인메모리라 할 수 있고, 다른 하나는 OS 측면으로 본 것으로써 프로그램이 적재되고 실행되어질 수 있는 부분이라고 본다. 그렇다면 HGC의 비디오 램은 메인메모리에 존재한다. 이러한 방법을 메모리 맵방식이라 말한다.

그러나 일반적으로 IBM/PC에 있어서는 HGC의 비디오 램을 메인메모리라 부르지 않는데 이건 잘못된 판단이다. 앞의 두가지의 개념을 모두 만족하는 부분임에도 불구하고 이 HGC의 비디오 램을 메인메모리라 칭하지 않는다는 것은 문제가 있다. 분명히 이 비디오 램에서도 프로그램을 실행시킬 수 있다. MS-DOS는 이 문제를 직접 지원하지 않지만 *.COM화일을 비디오 램에 로드시키고 점프명령을 사용하여 점프하면 아무 무리가 없이 프로그램이 수행된다(**그림 6 참조**).

약간은 주제를 벗어난 느낌이 드는데 여하튼 HGC의 비디오 램은 메인메모리내의 한 부분에 존재함에는 틀림이 없다. 정확한 번지는 그림 6에서 볼 수 있듯이 B0000H부터 BFFFFH까지로 64KB이다. 이 부분은 8086 프로세서가 직접 액세스를 할 수 있는 메인메모리영역이다. HGC는 래스터 스캔방식을 사용하는 비디오 어댑터라 하였기 때문에 비디오 램을 필요로 한다는 결론을 얻었고 그 영역을 번지로 나타내었다.

그럼 이제 HGC의 비디오 램이 어떤 포맷으로 이용되는지를 알아보자. HGC

〔그림 5〕 비디오램과 주사선 이동경로

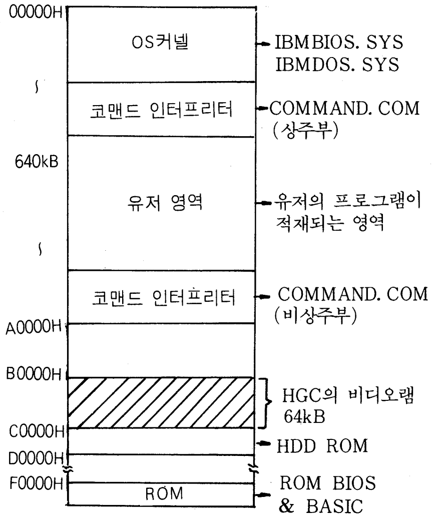



에는 두가지의 동작모드가 있는데 하나는 텍스트모드이고 또 하나는 그래픽모드이다. 텍스트모드란 아스키코드와 컬러 특성코드만 넣어주면 해당문자를 HGC가 화면에 뿌려주는 모드로 보통의 프로그램을 동작시킬 때에 머물러 있는 곳이다. 이 텍스트 모드에서는 화면에 대응되는 문자코드와 컬러코드만을 필요로 한다. HGC는 가로 80자, 세로 25자를 표시할 수 있으므로 80*25*(아스키코드 1바이트+컬러 코드 1바이트), 4000바이트의 텍스트 버퍼(비디오 어댑터가 스캔을 하는 메모리)가 필요하게 되고 그 번지는 B0000H부터 4000 바이트이다. 짝수번째의 바이트는 해당위치의 문자에 대한 아스키 코드값이고 홀수번째의 바이트는 해당 위치의 문자에 대한 컬러 특성코드 값이다.

그러나 HGC는 단색 어댑터이므로 이 코드를 컬러의 가지수로 표현하지 않고 갖가지 특성을 가자는 문자로 표시를 한다. 예를 들어 깜박거리는 문자라든지 밑줄이 쳐지는 문자, 밝은 문자, 반전된 문자 등등이다. 만약 B0000H 번지에 65라는 값을 넣고 B0001H 번지에 112라는 값을 넣는다면 **그림 7**과 같이 화면 제일 왼쪽 위에 반전된 대문자 A가 나타날 것이다. 이 번지는 차례대로 대응되어 있으므로 계산은 어렵지 않다.

만약 맨 왼쪽 맨 아래에 반전된 대문자 A를 찍고 싶다면 그 번지 계산은

〔그림 6〕 IBM/PC의 메모리 맵




24*160+0=3840으로 B000H부터 3840만큼 떨어진 바이트에 65를, 그보다 한 바이트 더 떨어진 바이트에 112를 써넣으면 된다. 이 112란 정해진 컬러특성 코드로 그 표는 HGC 매뉴얼에 나타나 있다.

여기서 24는 제일 아랫줄이 24줄을 건넌 25줄째이기 때문에 나온 값이고 160은 한줄에 80자이므로 160바이트를 필요로 하기 때문에 나온 값이다. 0은 첫번째 칸이기 때문에 얻은 수치가 된다. 각자 계산해보라.

또한가지의 동작 모드가 바로 그래픽 모드이다. 이 모드는 화면의 모든 점들을 일일이 제어할 수 있는 모드로써 그래픽을 표현하고자 할 때에 사용

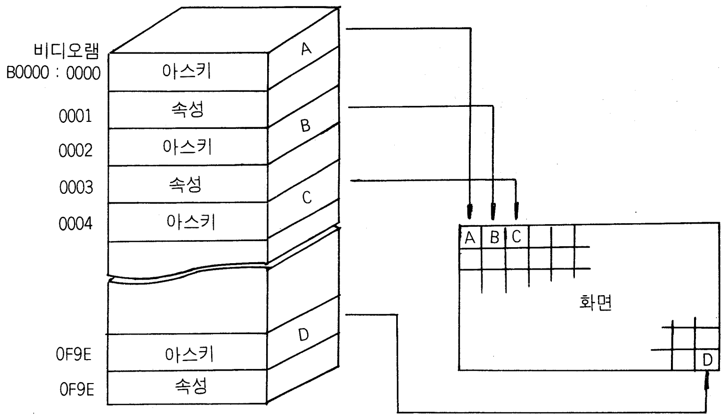
된다. HGC의 표준해상도는 720＊348이다. 단색 어댑터에서는 한 점을 표현하기 위하여 하나의 비트만을 할당하면 된다. 그 비트가 1이라면 해당 좌표의 점은 밝게 빛날 것이며 그 비트가 0의 값을 가진다면 점은 화면에 나타나지 않을 것이다. 따라서 HGC를 그래픽 모드로 전환시키고 비디오 램에 데이터를 적절히 써 넣으면 그래픽 화면을 **그림 8**과 같이 구성할 수 있다. 점 하나에 1비트가 대응되고 해상도가 720＊348 이므로 이를 곱해 8로 나누면 1 페이지분의 비디오 램의 크기를 구할 수 있는데 정확하게 32KB이다. HGC의 비디오 램은 64KB라 하였으므로 결과적으로 2페이지 분의 버퍼가 된다(**그림 9** 참조). 따라서 HGC는 두 장의 그림을 비디오 램에 올려놓고 번갈아 보여 주는 기능을 구현할 수 있다. 그에 대한 세세한 함수는 이 주제와 걸맞지 않으므로 생략하기로 한다.

그래픽 모드에 대한 설명을 정리하면, B0000H부터 32KB가 하나의 페이지에 대한 비디오 램이 되고 그 부분에 적당한 데이터를 기록하여 화상을 만든다고 하였다. 실지로 그래픽 모드에서 B0000H번지에 80H를 써 넣으면 맨 왼쪽 위 모서리에 점이 하나 나타나게 되는데 그 이유는 80H가 HGC에 의하여 디코딩 되어져 점으로 표현되었기 때문이다. 정확한 HGC의 사용방법을 알고자 하는 독자는 관련 매뉴얼을 찾아보기 바란다. 이 기사가 HGC를 프로그램 하는 방법을 제시하는 란이 아니므로 그 부분은 생략하겠다.

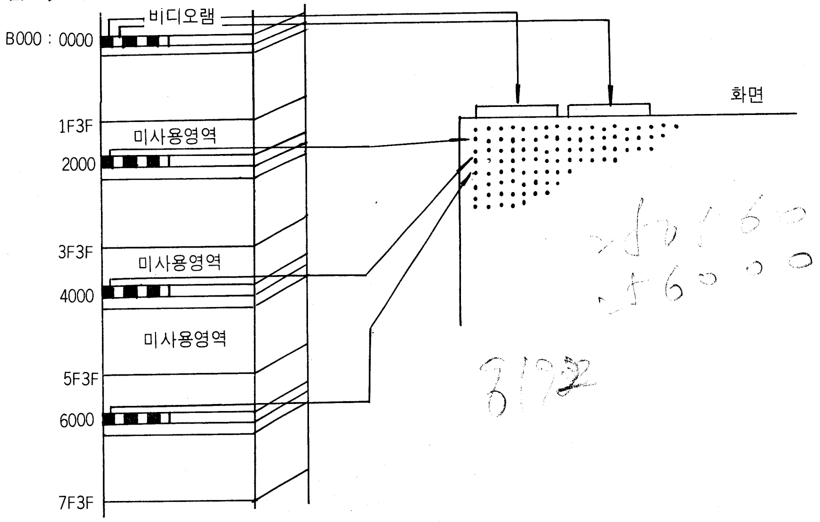
다시한번 HGC를 정리하면, 이는 래스터 스캔방식의 어댑터이고 따라서 비디오 램을 가지는데 그 위치는 PC의 메인 메모리영역이다. 이 HGC는 두가지 동작모드를 가지는데 첫번째가 텍스트모드, 두번째가 그래픽 모드이다. 텍스트 모드는 그림 7과 같이 아스키 코드와 컬러 특성코드를 텍스트 버퍼에 넣어줌으로써 문자를 화면에 표시하는 모드이고 그래픽 모드는 그림 8과 같이 화면의 각 점에 대응하는 비트를 설정하고 해당 비트를 콘트롤하므로써 화상을 만들 수 있다고 하였다.

이젠 그래픽 어댑터에 대한 이해를 어느정도 하였으리라 믿는다. 그중 우리가 흔히 접하는 HGC를 일례로 들었는데 일반적인 어댑터들이 이와 비슷한 방식을 사용하고 있다. 따라서 이러한 개념만을 확고히 하고 어댑터의 어플리케이션에 들어간다면 매뉴얼을 옆에 두는 것 만으로도 성공적인 결과를 얻을 수 있을 것이다. PC

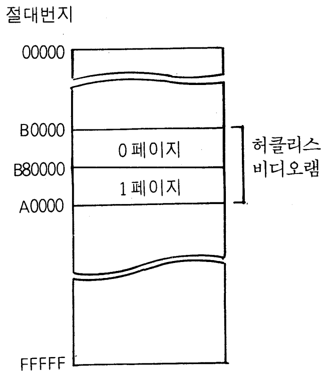
〔그림 7〕 텍스트모드와 비디오램




〔그림 8〕 허큘리스 그래픽모드(640×400)와 비디오램




〔그림 9〕 허큘리스카드의 그래픽 페이지



# 둘리가 본 컴퓨터세계 5
# 컴퓨터와 속성연애

이만희/ 한국교육개발원 컴퓨터교육연구센터

## 가까이, 조금 더 가까이

오래전 큰 맘 먹고 컴퓨터를 장만해 두었지만 그동안 별로 사용하지도 못하고 있는 사람은 지금쯤 들어간 밑천을 억울해하면서 방 한구석에 처박아 두고 먼지를 씌우면서 구박을 하고 있을 것이다. 또는 이제 막 컴퓨터를 들여온 사람중에는 본래 자신이 기대했던 현명하고 싹싹한 컴퓨터의 모습과는 딴판으로 우둔하고 무정한 모습에 점차 처음 사로잡혔던 열정이 식어가는 이도 있을 것이다.

더우기 사장님의 과감한 결단만으로 실무자가 준비할 틈도 없이 컴퓨터를 들여온 회사도 있다. 이렇게 되면 싫어도 억지로 컴퓨터 앞에 앉아서 일하는 척해야 하는 말단 직원은 컴퓨터가 오히려 괘씸하기 조차 할 것이다.

답답한 사람 중에는 서점에 찾아가 컴퓨터 활용의 기초에 관한 서적을 사서 열심히 보기도 한다. 그러나 대부분의 경우 여러가지 바쁜 일이 자꾸 생겨 계속 시간을 내서 공부하기도 어려워지고, 또 컴퓨터 프로그램이 그렇게 호락호락 쉽게 작성되지도 않는다. 실제로 업무에 사용될 만한 훌륭한 프로그램을 작성할 수 있는 사람이라면 이미 오랜 기간 컴퓨터를 다루어 온 프로그래머가 대부분이다. 오래전에 학교를 졸업해서 컴퓨터를 배운적이 없거나 지금 당장 컴퓨터를 사용해야 하는 급한 처지의 사람이라면 컴퓨터가 가까이 하기엔 너무나 멀리있는 존재로 느껴질 것이다.

이처럼 컴퓨터와 아직 서먹서먹한 사람이라면 골치아픈 프로그래밍은 나중에 배우고 우선 컴퓨터와 일을 함께 하면서 가까와지는 방법을 찾아야 한다.

이럴 때 워드프로세서, 스프레드시트, 데이터베이스 중에 한가지 만으로도 컴퓨터와 함께 생활하는 즐거움을 마음껏 누릴 수 있다.

이미 컴퓨터를 사용하려고 노력했다가 무참히 실패하고 컴퓨터와 남남이 된 사람도 있을 것이다.

"가까이, 조금 더 가까이" 이것만이 컴퓨터를 사랑스런 애인으로 만들 수 있다.

## 변덕장이를 위한 워드프로세서

### (1) 갈고 더 닦아서

말, 우리는 생활에서 서로의 의사를 전달하기 위해 수많은 말을 한다. 생각을 글자로 바꿀 때는 힘이 더 들고 그 결과가 오래 남기에 깊이 생각하고 적는다. 누구나 서류를 작성하거나 글을 쓸 때는 여러번 고치고 다시 쓰고 최선을 다해 마침내 만족한 단계에 도달한 후에 깨끗한 종이에 정성을 다해 옮겨 적거나 타이피스트에게 부탁하여 산뜻하게 타이핑을 한다. 그런데 마지막으로 완성된 문서를 살펴보다 갑자기 중간에 꼭 들어가야 할 말이 생각나거나 또는 잘못된 문장이 발견되어 그것을 지워야 한다면, 다음 두가지중 하나의 괴로운 선택을 해야 한다.

첫째 방법은 전부 새로 작성하는 것으로 이때는 손목에 고통이 느껴지거나 타이피스트에게 변명할 말을 생각해야 한다. 둘째는 고치는데 들어가는 고통은 크지만 결과는 그만큼 가치가 없다고 생각하여 미진한 양심을 덮어둔채 그냥 지나가는 것이다.

그러나 컴퓨터를 이용하게 되면 이런 어려움을 간단히 해결할 수 있다. 이 해결책이 워드프로세서로 문서에서 한글자만 틀려도 화를 내거나 변덕이 심해서 자주 계획을 바꾸어 다시 자료를 만들어 올려야 하는 높은 분을 모시고 있어도 웃으면서 일할 수 있다.

### (2) 머리속의 생각에서 컴퓨터 화면의 문자로

"워드프로세서" 우리 식으로 부른다면 "말틀" 사람이 사용하는 말을 컴퓨터 화면으로 보면서 처리하도록 해주는 소프트웨어로서 컴퓨터 사용자의 60%가 워드프로세서로 사용하고 있을 만큼 모든 컴퓨터에서 가장 많이 사용되고 있다.

영어의 "Word Processor"라는 이름이 가리키듯이 "말 처리기"라는 뜻이 되지만 워드프로세서가 처리하는 범위는 단순히 단어를 처리하는 수준을 넘어서 사람의 생각자체를 보다 창조적으로 발전시키는 단계까지 그 역할이 확대되고 있다. 워드프로세서는 아이디어 프로세스(Idea Process), 워드프로세스(Word Process), 데이터프로세스(Data Process) 등의 역할을 담당

하고 있다.

① 아이디어 프로세스

사람의 오랜 역사에서 생각을 표현하는 문자가 발명된 후에 문자의 중요성이 강조되어 관리의 등용이나 심지어는 사람의 학식과 인품마저도 그 사람의 문장으로써 평가하여 왔다. 그러나 문자를 처리하는 형태는 큰 변화가 없었다. 즉, 사람이 머리속으로 생각한 후에 손으로 쓰는 형태는 이미 수 천년전에 만들어진 붓을 비롯하여 그후 개선되어 발명된 펜촉, 만년필, 연필, 볼펜이 나온 후에도 마찬가지였다. 즉 문장의 처리는 사람의 머리 속에서만 가능하며, 필기도구는 단지 그것을 옮겨 적는 역할만 할 뿐이었다. 좋은 볼펜을 사용하여 글 솜씨가 나아졌다는 말을 들은 적이 있는가?

그러나 이제 "워드프로세서"가 바로 이런 형태-지난 수 천년간 문장을 작성하는 작업에서 변함이 없었던 -의 사고 능력을 갖고 있는 사람의 두뇌 속에서만 이루어져 왔던 것을 변화시키고 있다.

나날이 값이 싸지는 컴퓨터에서 사용할 수 있는 워드프로세서의 보급은 보통사람의 평범한 생각 즉, "말"을 처리하게 되었다. 컴퓨터 혁명은 몇 몇 특별한 과학자들의 어려운 계산을 도와주는 역할에서가 아니라 보통사람의 평범한 문장작성에 사용됨으로써 진정한 의미로서 역사의 변화를 가져오고 있다.

문장을 작성하는 일을 분석해보면 먼저 적합한 단어를 선택하고 순서에 맞게 배열하여 연결하는 말을 붙이는 작업이다. 때로는 영감으로 문장을 만들어 낸다고 하지만 그 이전에 무의식중에라도 사람의 머리속에서는 수많은 단어들이 흐르면서 생각을 이루는 작업이 진행되어 어느 순간 죽순이 땅을 뚫고 올라오듯이 언어로 나타나는 것이다. 이와 같이 적절한 단어를 고르는 작업은 간단한 것 같으면서도 정말로 어려운 일이다. 위대한 시인조차 적당한 단어를 고르지 못해 뼈를 깎는 고뇌를 하는 경우가 얼마나 많은가?

아직 한글화는 않되겠지만 발전된 워드프로세서는 컴퓨터 화면에 여러 동의어들을 보여주기 때문에 적절한 단어를 선택하는데 도움을 주고 있다.

② 워드프로세서 : 자료의 변경

타자기가 발명되어 글씨 모양이 형편없는 사람들의 컴플렉스는 감출 수 있게 되었지만 이것 역시 한번 찍힌 글자는 바꾸기가 어렵다. 한두자 바꾸는 정도야 타자 수정액을 쓰거나 콜렉션 리본으로 가능하지만 몇줄을 새로 끼워 넣거나 또는 몇페이지에 걸쳐 내용의 일부분이 밀리거나 옮겨가는 일이 생긴다면 두손을 들어 버리게 된다.

또한 어느 회사에서 부장님이 산으로 야유회를 갈 계획을 수립하라고 지시해서 담당 직원이 몇일을 걸려 계획서를 다 작성한 후에 갑자기 사장님이 야유회는 바다로 가는 것으로 마음을 바꾼다면 담당자는 허탈해 지지만 당면한 문제– 계획서를 당장 다시 작성해야 하는 것–를 해결해야 한다. 이럴땐 차라리 사장님을 어떻게 해서든지 설득해서 산으로 갈 수 없을까 생각하기 마련이다.

그러나 사실 산으로 가거나 바다로 가거나 야유회 계획서의 형태는 큰 변화가 없다. 참석인원, 비용, 소요일자 그 모든 것이 대부분 동일하다. 단지 일부 내용을, 산에서 바다로, 등산장비를 수영복으로 바꾸기만 하면 그 계획서의 대부분은 그대로 쓸 수가 있다. 이때 능숙하게 워드프로세서를 사용한 사람이라면 불과 몇분후에 수십페이지의 야유회 계획서를 다시 작성해서 올릴 수 있다.

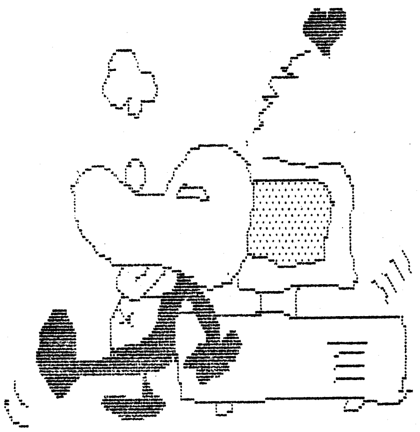
이와 같이 워드프로세서는 한번 작성한 문서를 다시 불러서 바뀌는 부분만 쉽게 수정하여 바로 사용할 수 있다.

3) 데이터프로세스

전산화를 추진하는 회사에서 컴퓨터를 모르는 일반 직원은 그가 하는 일이 너무 바쁘다 보니 이제까지 해오던 일이 정리되지 않고 있어 컴퓨터에 입력이 불가능하다고 단정해 버리고 스스로가 수치스럽게 느껴 컴퓨터 전문가에게 담당업무를 잘 알려주지 않을 수 있다. 그러나 그의 머리속에는 이미 하고 있는 일을 사용하기 편리하게 정리하고 있다. 그가 자재를 관리하는 직원이라면 대강 어느 부속이 어느 곳에 몇개 정도 있고 얼마후에 보충해야 한다는 것을 알고 있다.

컴퓨터에 대한 지식이 없는 그 직원도 컴퓨터에 대한 지식없이 바로 사용할 수 있는 워드프로세서는 그의 머리속에서 들어 있는 지식이나 약간 지저분하지만 그 나름대로 정리해온 장부나 노트를 쉽게 정리해 줄 수 있다.

또 단순히 깨끗하게 정리하는 것만이 아니라 거기에 입력된 자료를 일반적인 컴퓨터 프로그램에서 그대로 이용할 수 있다. 컴퓨터에 자료를 입력하기 위해 별도의 프로그램을 작성해서 사용하는 경우가 많지만 워드프로세서 만큼 손쉽게 입력하기가 어렵다. 대규모 통계처리를 위한 수많은 자료도 일단 워드프로세서로 입력하면 처리가 간편해진다. 컴퓨터 프로그램을 작성하는 일도 요즘은 워드프로세서를 많이 이용한다. 워드프로세서는 프로그램 언어가 갖고 있는 에디터를 사용하는 것보다 손쉽게 입력할 수 있다.



## (3) 워드프로세서의 발달과 구성

### ① 워드프로세서의 역사

워드프로세서는 컴퓨터의 에디터(Editor) 에서 시작되었다. 에디터는 컴퓨터에서 사용할 프로그램을 작성할 때 입력을 편하게 하고 틀린 부분을 손쉽게 고칠 수 있도록 하기 위해서 개발된 것이다. 요즘은 거의 사용되고 있지 않지만 초기의 컴퓨터는 카드에 구멍을 뚫어 한장에 한줄씩 프로그램을 기록했다. 프로그램을 고치려면 몇장의 카드를 새로 끼워 놓거나 바꾸는 방법을 사용했다.

그후 TV 화면과 같은 컴퓨터 디스플레이 장치가 개발되자 종이 카드대신 화면에 한줄씩 프로그램을 나타내고 고치는 "라인에디터"가 등장했다. 이때 컴퓨터 화면에 커서가 보이는데 커서는 연필이 종이에 닿고 있는 부분과 같은 것이다. 즉, 종이 위에 연필로 글자를 적는다면 연필끝이 위치한 곳에 글자가 써지듯이 컴퓨터에서는 커서가 있는 곳에 글자가 나타난다. 라인에디터는 아직도 사용되며 베이직에서 커서를 움직여 한줄씩 고치고 입력하는 것등이 이와 유사한 기능이다.

이 라인에디터의 기능이 더욱 개선되어서 한줄 단위로 프로그램을 고치고 입력하는 것이 아니라 커서가 화면을 마음대로 옮겨 다니면서 작업할 수 있는 화면에디터 즉, "스크린에디터"가 개발되었다. 프로그램을 작성하다보면 나중에야 생각이 나서 앞으로 다시 돌아가 고쳐야 할 일들이 자주 생기기 마련이다. 이 스크린에디터를 프로그램 작성에만 사용할 것이 아니라 우리가 평소에 하는 말이나 글을 작성하는 일에도 사용한다면 컴퓨터 화면에서 마음대로 쓰고 지울 수 있는 편리한 워드프로세서가 되는 것이다.

더우기 컴퓨터 화면은 종이와는 달리 몇번을 쓰고 지워도 버리는 휴지가 생기지 않기 때문에 마음놓고 이렇게 저렇게 내용을 바꿀 수 있다. 또 화면 글자체가 보기에 그렇게 흉하지 않아서 손으로 쓰는 경우보다 읽기가 편하다. 이렇게 워드프로세서가 일단 사용되기 시작하자 보다 개선된 기능을 갖고 있는 다양한 워드프로세서가 속속 개발되어 이제는 수백가지의 워드프로세서가 나와 치열한 경쟁을 하고 있다.

### ② 워드프로세서의 구성

워드프로세서의 종류는 수백가지가 있다보니 그 기능이 각기 다양하지만 기본적인 기능은 입력기능, 편집기능, 출력기능 등으로 구성된다.

**• 입력기능**

타자기와 같은 기능으로 컴퓨터 키보드를 이용하여 자료를 입력하는 것이다. 한줄의 길이를 정하는 워드랩 기능이 있어 한줄이 다 채워졌을때 자동적으로 다음 줄로 바뀌어 입력이 되므로 한줄이 넘지 않을까 염려할 필요가 없다.

자주 사용하는 단어나 문장, 도형같은 것은 키하나(기억키 : Macrokey)에 기억시켜 필요한 위치에서 그 키하나만 누르면 기억된 단어나 도형이 자동적으로 입력된다.

한자의 사용에 있어서는 한글과 똑같은 발음의 한자를 화면에 보여주면서 그중에 맞는 것을 골라 정확한 획을 몰라도 대충 읽을 수만 있으면 한자를 사용할 수 있다.

서류를 작성하다보면 도표나 그래프, 도형을 그려야 하는 경우가 많다. 이런 작업을 전문으로 하는 프로그램도 있지만 워드프로세서 중에는 상당한 수준의 그림이나 그래프 등을 그릴 수 있다.

또한 입력장치중에 마우스를 이용하면 복잡한 그림을 쉽게 그릴 수도 있고 스캐너를 이용하면 사진이나 삽화 등도 실물 그대로 간단히 입력할 수 있다. 물론 그림의 크기나 방향도 문서내에서 알맞게 변화시킬 수 있다.

**• 편집기능**

워드프로세서는 한번 입력한 자료를 필요에 따라 다시 불러서 마음대로 고칠 수 있다는 것이 최대의 특징이며 장점이다. 따라서 워드프로세서로 팔리는 상품의 품질은 편집기능이 얼마나 우수한가에 달려있다. 워드프로세서가 사람의 생각을 도와주는 아이디어프로세싱을 할 수 있는 것은 편집기능이 있기 때문이다.

이 편집기능은 입력한 단어나 문장

〔화면 1〕 프린터 출력 예

| | |
|---|---|
| 옆으로 두배 확대 | Example |
| 위아래로 두배 확대 | Example |
| 진하게 | Example |
| 역상 | Example |
| 음영 | Example |
| 밑줄 | Example |
| 그림자 | Example |
| 외각선 | Example |

의 자리에 새로운 내용으로 다시 입력하는 것으로 원하는 곳에 커서를 옮겨놓고 새로 입력하면 그 자리에 먼저 있던 내용은 감쪽같이 지워지고 새로 입력한 것으로 바뀌게 하고, 입력된 문장 사이에 추가하여 단어나 문장을 끼워 넣기도 하며 필요없는 부분을 지우기도 한다.

또한 내용의 일부를 다른 위치로 옮길 수도 있고, 똑같은 내용을 다른 곳에 복사할 수도 있으며, 입력된 내용중에 원하는 글자나 단어를 찾아 볼 수 있다.

그 외에 특정한 글자를 다른 글자로 대치하며, 자체적으로 사전을 보유하고 있어 철자법이 틀린 낱말을 표시하여 주고 입력이 고르게 입력되지 않았다면 그 문장에 관한 정렬기능이 있어 보기좋게 해준다.

• **인쇄기능**

화면에 표시하는 문자의 크기를 마음대로 조절하기는 어렵지만 프린터로 종이에 인쇄할 때는 옆이나 아래위로 확대하여 인쇄하는 것이 가능하다. 프린터 내부에도 마이크로프로세서가 들어있기 때문에 여러가지 특별한 효과를 내어 인쇄할 수 있다.

특정부분을 강조하기 위한 밑줄긋기(Underlining)나 두번찍기(Double strike) 또는 수학이나 화학식에 사용되는 작은글자(Subscript)도 인쇄가 가능하다(화면 1 참조). 이외에도 페이지 숫자 자동인쇄 등 인쇄에 따른 다양한 기능이 많다.

③ 워드프로세서의 방향

컴퓨터 세계로 안내하는 길잡이와 같은 워드프로세서는 단순히 입력하고 수정하는 범위를 넘어서 기능에 있어서 비약적인 발전을 하고 있다. 이중에 몇가지 중요한 추세는 다음과 같다.

• **아이디어프로세서 기능의 확장**

편집기능의 향상은 우리의 생각을 도와주는 아이디어프로세서의 역할을 강화시키고 있다. 검색과 사전기능은 철자법 확인이나 동의어만이 아니고 가사나 시를 쓸 때도 사용된다. 운을 맞추어 사용한다면 같은 운으로 끝나는 단어를 화면에 보여주고 고르며 반대로 첫음절이 같고 다음에 끝나는 음절이 다른 단어를 보여 준다든가하는 상상하기 어려운 높은 수준의 예술의 경지에까지 이르고 있다.

• **조작의 편리**

워드프로세서의 수많은 기능을 다 머리속에 기억하면서 사용해야 한다면 실제로 일하기가 불편해진다. 선택할 수 있는 기능을 설명하는 메뉴를 보여주고 고르는 것과 화면 한구석에 기능을 상징하는 조그만 그림으로 선택하기만 하면 되는 아이콘기능 등 사용자가 편리한 입장에서 사용할 수 있는 워드프로세서도 나오고 있다.

• **책상 출판**(DTP : Desk Top Publishing)

워드프로세서의 인쇄기능과 편집기능의 향상은 책상위에 놓은 한대의 컴퓨터와 프린터만 갖고도 멋진 책의 원판을 인쇄할 수 있는 단계로 가고 있다. 물론 고급출판 전용의 워드프로세서를 이미 많은 출판사에서 책을 만드는데 사용하고 있지만 레이저 프린터의 등장으로 개인이 혼자서도 책상에 앉아 책을 만드는 단계에 접근하고 있다.

## 의심 많은 사람은 스프레드 시트

### (1) 컴퓨터를 잘 팔리게 해준 프로그램

집에 값비싼 오디오를 구입한다면 그 이유는 무엇인가? 그것은 당신이 좋아하는 가수의 노래가 담긴 테이프나 디스크가 있기 때문이다. 만일 들을 만한 노래가 없다면 오디오를 살 이유가 없다. 마찬가지로 개인용 컴퓨터를 구입하는 이유도 자기가 하는 일을 도와줄 수 있는 프로그램이 있기 때문이다.

1979년 하버드 대학에서 경영학을 전공하는 대학원생인 댁 브레클린은 재정예측 과목에서 반드시 필요한 수많은 계산을 하는 편리한 프로그램을 생각해 냈다. 이것이 스프레드 시트의 시초가 된 "Visicalc"였다. 그는 종이에 계산표를 그리는 것과 똑같이 컴퓨터 화면위에 상하 좌우로 선을 그어 여러개의 직사각형 칸(Cell)을 만들고 각각의 칸은 행과 열로 그 위치를 구별하도록 하였다. 그리고 워드프로세서와 마찬가지로 커서를 마음대로 상하 좌우로 옮겨 다니며 현재 커서가 있는 위치에서 그 칸 하나 하나에 숫자를 적을 수 있도록 하였다.

이 프로그램의 가장 중요한 아이디어는 여유로 남아 있는 한칸에 앞서 입력한 숫자들을 계산하는 방식을 별도로 입력하면 그 결과가 즉시 계산되어 그 칸위에 표시되도록 하는 것이다. 휴대용 계산기와 별로 나아진 것이 없어 보이지만 중요한 차이는 앞서 입력한 숫자중에 하나를 바꾸어주면 자동적으로 다시 계산하여 변경된 결과가 표시되는 것이다.

또 단순히 덧셈 뺄셈만이 아니고 몇개의 칸들이 서로 연관을 갖고 곱하거나 나누어 계산이 되는 복잡한 식을 한번 입력하면 몇번이고 복사해 쓸 수 있다. 따라서 아무리 골치아픈 계산작업도 결과가 바로 눈앞에 보이기 때문에 자료를 여러가지로 바꿀 때 그것에 따라 변하는 결과가 흥미진진하게 기다려 지는 것이다.

이와 같이 편리한 기능을 발휘하는 프로그램이 널리 퍼지자 이 프로그램을 사용하기 위해 퍼스널컴퓨터를 구입하려는 사람도 많았다. 82년초 필자가 Visicalc에 관심을 갖게 된 것도 미국의 컴퓨터 잡지의 광고에서 컴퓨터 하드웨어의 우수성을 선전하고자 이 프로그램이 수행되는 것을 내세우는 것을 보고 무엇이 그렇게 좋은 것인가 궁금했기 때문이었다.

### (2) 스프레드 시트의 발전

① 로터스 1-2-3의 등장

스프레드 시트는 종이에 계산표를

그려서 계산을 하는 방식을 컴퓨터 화면으로 옮겨온 것으로 정확히 말한다면 컴퓨터 메모리에서 계산을 하고 그 결과를 화면에 보여주는 것이다. 때문에 컴퓨터 메모리를 종이처럼 사용한다는 의미에서 그 명칭이 유래된 것이다. 스프레드 시트의 시초인 Visicalc가 엄청난 인기를 끌자 이와 유사한 기능의 프로그램들이 나오기 시작했는데 멀티플랜이나 슈퍼캘크도 이중 하나이다.

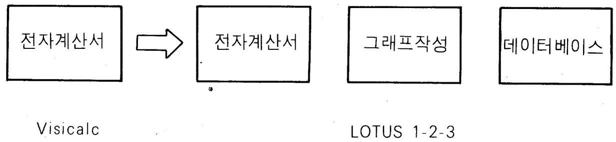
16비트 컴퓨터 시대가 되자 컴퓨터의 능력이 크게 증가하고 스프레드 시트도 계산표 작성에 이어서 보다 다양한 기능이 가해졌다. 사무실에서 계산표를 작성할 때 각종 그래프를 그리는 경우가 많다. 또 자료를 이름순서나 크기순서로 정리하기도 한다. 이런 일들을 프로그램 하나로 모두 할 수 있는 스프레드 시트인 "로터스 1-2-3(Lotus-)" 가 등장했다. 계산서 기능, 그래프 기능, 데이터베이스 기능 이렇게 세가지 기능을 갖고 있다는 뜻으로 1-2-3라고 불려져 여러가지 스프레드 시트중 압도적인 인기를 차지하게 되었다(그림 1 참조).

심포니(Sympony)나 프레임 워크(Framework) 등 보다 많은 기능을 갖고 있는 것이 발표되어도 새로운 것을 또 배우는 것이 귀찮은 사람들이 많아서 그 확고한 위치가 변하지 않고 있다. 사무실에서는 다른 어떤 도구 보다도 많이 사용되어 신입사원을 선발할 때 로터스 1-2-3 사용능력을 기본 자격으로 요구하는 회사도 늘고 있다.

**② 농구장보다도 넓은 종이 위의 작은 창문**

스프레드 시트는 컴퓨터 메모리를 종이처럼 사용한다. 스프레드 시트인 로터스 1-2-3는 화면에 그리는 직사각형 칸의 크기를 원하는대로 늘이거나 줄일 수 있기 때문에 전체 크기를 얼마라고 단정할 수는 없다. 그러나 한 칸을 가로 1㎝, 세로 5ｍ라고 생각하면 옆으로 256열 아래 위로 8192행의 계산표이므로 넓이 13m, 길이 81㎝가 된다. 이렇게 농구장보다 더 커다란 계산용지를 본적이 있는가? 여기에 숫자를 입력하고 마음대로 계산할 수 있는 200만개도 넘는 칸이 준비되어 이렇게 넓은 종이를 한눈에 다 볼 수는 없으므로 컴퓨터 화면에 보이는 것은 극히 작은 일부분이다. 컴퓨터 화면은 작은 창문과 같이 메모리의 한부분만을 들여다 보면서 계산을 하는 것이다.

〔그림 1〕 스프레드 시트의 발전과정




**③ 세상의 모든 것을 세가지로**

**• 자료의 종류**

로터스 1-2-3는 계산을 주로 하는 입장에서 모든 자료를 다음 세가지로 구분한다. 첫째는 숫자(Number) 자료로 갯수나 가격 등 실제 크기를 나타내는 값으로 0부터 9, 소숫점, ＋, － 기호만 입력되고 최대 240자의 길이까지 입력할 수 있다.

둘째는 문자(Label)자료로 각항목의 이름에 해당하는 것으로 레이블은 보통 계산서식의 가로와 세로에서 항목의 이름을 표시하기 위해 사용된다. 따라서 첫글자는 문자로 시작되며 레이블은 최대 240자까지 입력할 수 있다.

세째는 수식(Formulas)으로 숫자를 계산하는 방식이 들어 있다. 계산식을 입력할 때 위치를 나타내는 B1을 그대로 입력하면 문자가 되므로 맨앞에 ＋나 $기호를 입력하여 ＋B1이라고 하면 문자 B1이 아닌 B1에 들어있는 값, ＋B1＋B2는 B1값과 B2의 값을 더해서 표시한다는 뜻이 된다.

**• 자동 재계산**

이 기능은 계산결과가 아닌 계산방식을 입력해 놓으면 숫자값이 아무리 변해도 자동적으로 결과가 계산된다. 이 계산식은 스프레드 시트에서 가장 중요한 부분으로 실제 업무나 회계장부에 사용되는 복잡한 계산양식도 한번만 만들어 두고 커서를 가져가서 새로운 숫자를 입력만 하면 계산식에 의해 순식간에 새로 계산을 한다.

**④ 명령어**

**• 명령어 선택**

스프레드 시트는 베이직과 같은 일반 컴퓨터언어와 달리 여러가지 명령어를 콘트롤 판넬에 보여주면서 그중에서 필요한 기능을 커서를 움직여 저장하거나 원하는 명령의 첫글자를 직접 입력하면 바로 원하는 작업을 할 수 있다. 명령어를 선택하기 위한 첫번째 작업은 ／키를 누르면 된다.

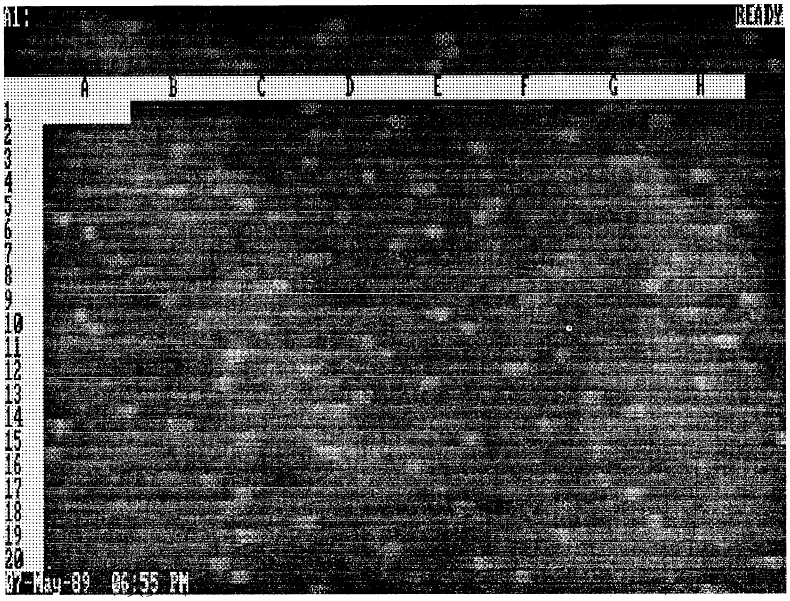
화면의 윗부분에 있는 콘트롤 판넬에 기본적인 명령들이 나타나고 막대모양의 커서가 그중 하나를 가르키고 있다. 콘트롤 판넬에 표시되는 명령어는 원하는 작업을 뜻하는 영어단어로 되어 있어 사용이 편리하고 기억하기도 쉽다.

**• 명령어의 계층구조**

로터스 1-2-3의 명령어는 아주 다양하다. 따라서 명령어를 모두 다 보면 오히려 혼란이 일어나므로 피라밋과 같은 구조로 되어 있다. 계산표를 만드는 작업을 비롯해서 프린트하고, 작업한 결과를 디스크에 보관하고, 그래프를 그리는 등 여러가지 다양한 명령이 있다.

화면 2과 같이 Worksheet 명령어 하나에도 많은 명령이 다음 단계에 준비되어 있어 기가 질리는 사람도 있겠지만 로터스 1-2-3는 명령어의 대부분을

〔화면 2〕 로터스 1-2-3 워크쉬트




몰라도 원하는 일을 하는데 지장이 없으므로 사용하면서 점차 익히면 된다.

⑤ 그래프 기능

로터스 1-2-3는 숫자의 계산 뿐만 아니라 입력하거나 계산한 결과로 다섯가지의 그래프를 그릴 수 있다. 또 숫자가 바뀌면 그래프도 자동적으로 바뀐 값으로 고쳐져 그려진다. 막대그래프, 누적막대, 꺽은선 그래프 , 원그래프, 산포도 등을 그릴 수 있으며 하나의 그래프에 여섯가지의 자료를 동시에 그릴 수 있다. 새로운 버전중에는 그래프의 종류가 더 많다.

⑥ 데이터베이스 기능

자료가 많으면 관리하기가 어려워진다. 고등학교 졸업생 수백명이 동창회를 만들어 회비를 받고 장부를 정리한다면 스프레드 시트를 사용하는 것이 편리하다. 그러나 동창들에게 카드를 보낼 때 실수로 빠뜨리거나 주소가 틀리는 경우도 생긴다. 이때 이들을 이름이나 주소순으로 정리하여 인쇄하거나 원하는 자료를 찾아 낼 수 있다면 정말 편리할 것이다.

로터스1-2-3는 데이터베이스 기능이 있어 이름이나 주소 등을 크기별로 올림차순이나 내림차순으로 배열할 수 있고 특정한 조건이 맞는 자료만 골라 낼 수도 있다.

### (3) 숫자로 고민하는 사람을 위하여

필자는 수학을 싫어하는 편은 아니었지만 깨알같이 적힌 숫자를 끝없이 계산하는 것은 자주 틀리고 정말 질려버리는 사람이었다. 암산만으로 척척 계산할 수 있는 사람이 부럽기도 하였지만 스프레드 시트를 알고부터는 나의 한심한 계산실력이 부끄럽지 않게 되었다.

나의 가족중에 한사람이 증권회사에 입사하여 매일 계속되는 숫자계산에 지쳐있는 것을 보고 스프레드 시트 사용을 권했다. 처음에는 컴퓨트에 대한 두려움으로 망설였지만 마침내 눈앞에서 일어나는 요술과 같은 계산작업을 보고 그 누구보다 열심히 컴퓨터를 사랑하게 되었다.

회사에서 숫자계산이 맞지않아 다시 계산하느라 퇴근이 늦어진 경험은 없는가? 그런 분들은 반드시 스프레드 시트를 사용해보기 바란다. 수천개의 숫자가 있어도 계산결과가 의심스러울 때 키하나만 눌러주면 수초내로 전부 다시 계산해서 결과를 표시해 줄 것이다. 혹시 학교에서 성적표를 계산하는 일이 있다면 즉시 숫자를 입력하고 계산식을 이용해 결과를 확인해 보라.

우리 주변의 작은 자료부터 로터스 1-2-3를 이용하여 작업을 시작한다면 어려웠던 계산이나 보고서작성이 즐겁고 창조적인 일로 바뀔 것이다.

## 빠뜨리지 않는 데이터베이스

### (1) 하나를 알면 열 곳에서 사용한다.

우리가 살아가는 세상에는 계속 수많은 정보가 생기고 쌓여가기 때문에 그 수가 많아지면 그 모든 것들을 정리해서 사용하기가 어려워진다. 이때 컴퓨터를 이용해서 자료를 정리하고 보관할 수 있는 데이터베이스를 이용해 보자.

만일 당신이 비디오테이프 가게를 운영하고 있다면 아침부터 저녁까지 여러사람에게 테이프를 빌려주고 돌려받고 있을 것이다. 손님 중에는 처음 오는 사람도 있고 그동안 주소가 바뀌어 연락처를 새로 정리해야 하는 경우도 있다. 열심히 일하는 주인이라면 고객관리 카드를 만들어 손님이 빌려간 테이프의 목록을 적어 놓고 그 손님이 얼마나 자주 빌려 가는지 알아 볼 것이다.

더욱 부지런한 주인이라면 테이프를 중심으로 관리하는 카드도 만들어 놓고 그 테이프를 누가 빌려 갔는지, 몇번이나 빌려 주었는지, 수익률은 얼마인지 알고 싶을 것이다. 그러나 이렇게까지 하려면 너무 손이 많이 필요하여 포기할 수 밖에 없을 것이다. 또 손님이 테이프를 빌려간지 오래 되었지만 일일이 카드를 확인하기 어려워 빠뜨리거나 잃어버리는 테이프가 있을 것이다.

이때 데이터베이스를 사용한다면 이런 복잡한 자료관리를 간단히 할 수 있다. 데이터베이스에 조금만 익숙해지면 고객이 그동안 빌려간 테이프를 분석해서 손님의 취향을 알아보고 좋아할 새로운 테이프만을 골라서 소개하는 것도 가능하다.

데이터베이스의 가장 좋은 점은 한번 자료를 입력하면 그 자료를 여러가지 각도에서 이용할 수 있는 것이다. 즉, 손님이 빌려간 비디오테이프 기록은 즉시 그날의 매상실적과 그 테이프의 인기도와 수익율을 알아보는데 사용되며 손님의 대출관리 자료와 취향분석에도 이용된다.

또 반납이 지연되는 테이프 목록을 매일 뽑아내서 독촉장을 보내는데 이용할 수 있다.

## (2) 데이터베이스의 챔피언 dBASE

본래 데이터베이스는 대형컴퓨터에서 많은 자료를 관리하기 위해 개발되었다. 여기에 필요한 도구로서 DBMS (Data Base Management System)는 컴퓨터언어(COBOL, BASIC 등)에서 보조적으로 사용되었던 유틸리티였다.

퍼스널컴퓨터가 널리 보급되자 대형에서만 가능하던 DBMS를 쉽게 이용할 수 있도록 한 데이터베이스 프로그램이 나왔다. 이 중에서도 아쉬톤 테이트(Ashton-Tate)사의 dBASE Ⅱ가 1982년 발표되자마자 높은 인기를 끌어 소형 데이터베이스 시장을 주도하게 되었다. 그 후에도 수백종의 다른 데이터베이스 프로그램이 나왔지만 dBASE Ⅱ의 독점을 막지 못했다.

16비트 시대가 되어서도 dBASE Ⅱ의 후계자인 dBASE Ⅲ가 계속 대를 이어 시장을 독차지하면서 퍼스널컴퓨터의 dBASE라고 하면 dBASE Ⅱ, Ⅲ를 가르키는 정도로 되었다. 더우기 dBASE Ⅱ, Ⅲ가 많은 인기를 얻자 이것을 자세하게 설명하는 안내책자만도 수십권이 나와서 사용자의 입장에서는 선택의 여지가 없을 정도가 되었다. 이렇게 특정 회사의 소프트웨어 한가지가 모든 시장을 차지하는 챔피언처럼 되어버린 경우는 스프레드시트의 로터스 1-2-3와 데이터베이스의 dBASE Ⅲ가 있다.

## (3) 데이터베이스의 구성

dBASE는 많은 자료를 전문적으로 관리하는 것으로 잘 활용하기 위해서는 컴퓨터 자료관리에서 사용하는 기본적인 개념을 이해하는 것이 필요하다. 비디오테이프 대출에 사용하는 카드의 예를 들어 설명하겠다.

데이터베이스는 컴퓨터로 하는 것만이 아니라 상점에서 손으로 적어서 정리하고 있는 장부도 일종의 데이터베이스이다.

① 데이터베이스(Data Base)

테이프 대출 기록장과 고객관리 카드 두가지 장부가 있다면 이 두가지 장부가 모여서 하나의 데이터베이스를 구성한다고 할 수 있다.

② 화일(File)

손으로 작성하는 장부 한가지나 카드를 묶어 놓은 하나의 철이 컴퓨터에 입력되어 있다면 하나의 화일이 된다.

③ 레코드(Record)

고객카드에는 여러 항목이 적혀있다. 이름, 주소, 전화번호 등 고객에 대한 자료가 모여 있을때 고객 한명의 기록에 해당하는 한장의 카드가 하나의 레코드가 된다.

④ 필드(Field)

고객관리용 카드의 경우 기록하는 이름, 주소, 전화번호 등 하나의 항목을 필드라고 한다.

## (4) 사용방법

① 초보자와 전문가

잘 모르는 곳을 찾아온 초보자에게는 항상 옆에 붙어 다니며 친절하게 설명을 해주는 안내자가 고맙지만 그곳에 익숙한 사람에게는 오히려 귀찮은 존재가 된다. dBASE Ⅲ는 이 두가지 종류의 사람을 모두 만족시키는 융통성을 갖고 있다.

당신이 만일 초보자라면 dBASE Ⅲ는 화면에 항상 도움말을 보여주고 간단한 명령어 몇가지만으로도 즉시 필요한 일을 할 수 있으며 풍부한 레포트(Report) 기능을 이용하면 간단하게 멋진 보고서를 인쇄할 수 있다. 또 화면에서 바로 자료를 추가하거나 입력된 자료를 불러 수정할 수 있어 스프레드 시트와 비슷한 편리함을 느낄 수 있다.

또 당신이 어느정도 dBASE Ⅲ에 익숙해지거나 전에 BASIC이나 COBOL로 프로그램을 작성해본 사람이라면 dBASE Ⅲ 자체에 프로그램을 할 수 있는 언어가 준비되어 있다. 이때는 귀찮은 도움말은 화면에서 지워버리고 전문가답게 프로그램을 작성하여 수많은 자료들을 일사분란하게 움직이도록 지휘할 수 있다. dBASE에서 사용되는 명령어의 대부분이 작업내용을 의미하는 영어단어로 되어 있어 배우고 익히기 쉽게 되어 있다.

dBASE의 여러 장점 중에서도 가장 뛰어난 것은 하나씩 명령을 해주는 직접모드와 이 명령어들을 모아서 자동적으로 계속 수행시키는 프로그램 모드가 있다는 점이다.

이상과 같이 컴퓨터는 사람이 다루기에 따라 친구와 같이 친하게 지낼 수도 있고 낯선 사람처럼 어렵게 여겨질 수도 있을 것이다. 이러한 모든 것들은 각자의 손에 달려 있으니 이 기회에 컴퓨터와 친해질 수 있는 방법을 연구해 보길 바란다. PC

**□ 알림 □**

지금까지 5회에 걸쳐 게재되었던 본 연재는 이번호를 마지막으로 끝마치게 되었습니다.

그동안 좋은 원고를 집필해 주신 이만희님께 감사드리며 다음호부터는 좀더 새로운 내용으로 독자여러분을 찾아 뵐 것을 약속드립니다.

# 국내소식

### –과기처
## S/W 개발비 산정

과기처는 지난 5월 3일 소프트웨어 개발비의 일반적 원칙, 소프트웨어 개발공정의 업무범위, 기초 소요공수의 보정을 통한 실질소요공수 산출 등 10 개항의 개발비 산정기준을 확정 고시 했다.

이번에 고시된 소프트웨어 개발비 산정기준에서는 S/W 개발비를 각 공정별로 소요공수를 산출한 후 이에 대한 직접인건비·직접경비· 제경비 및 기술료의 합계액으로 산정함을 원칙으로 했다.

또 이 기준의 적용을 받는 S/W 개발공정은 요구분석·설계·프로그램 작성/통합테스트 및 설치, 유지보수로 구분하고 각 단계별 업무범위를 정했다.

소프트웨어 소요공수는 기초소요공수를 산정한 후 여기에 보정을 통한 실질소요공수를 곱해 산출하도록 하였으며 기초소요공수는 사무처리용 프로젝트와 비사무처리용 프로젝트 등으로 나누었다.

기존 소요공수에 곱해지게 되는 규모별 보정계수는 7만~15만 스탭을 1로해 15만~30만은 1.1, 30만~50만은 1.3, 50만~70만은 1.5, 70만~1백만은 1.7, 1백만 이상은 2로 했다.

이밖에 프로젝트 형태별 보정계수는 사무처리용 소프트웨어를 1로해 과학기술용 소프트웨어 1.5, 시스템용 소프트웨어 2.4, 기종이 다른 시스템간의 접속을 위한 통신용 소프트웨어 3, 공정제어용 소프트웨어 3.2, 지휘 및 통제 소프트웨어 5.6 등으로 정해졌다.

### –한국전기통신공사
## 국산전자교환기 TDX-1B 개통

한국전기통신공사의 통신장비 국산화 계획에 따라 추진해온 전자교환기 개발이 각업체들과 공동개발의 성과로 국내 전자교환기사업의 기술축적 및 수입 대체효과를 가져왔다.

국내기술진에 의해 개발된 중용량 전자교환기인 TDX-1B가 지난 4월 25일 개통됐는데, 이는 지난 86년 TDX-1이 개발되어 상용화된데 이어 또 하나의 개가인 동시에 정보화 사회의 터전을 이룬 쾌거로 평가되고 있다.

이번에 개통된 TDX-1B는 통신공사 및 전자통신연구소와 금성반도체, 대우통신, 동양전자통신, 삼성전자, 동아전기 등 5개사가 지난 86년부터 공동개발을 하였으며, 용량은 2만2천5백28 회선 규모의 중용량 전자교환기이다.

한편, 전기통신공사는 이번 TDX-1B 개통을 시작으로 올해안에 모두 36만 7천회선을 개통키로 했으며, 내년에는 이보다 대폭 늘어난 1백21만9천6백회선을 설치하는 등 국설교환기의 주력 공급기종으로 부상시킬 계획이다.

TDX-1B는 기존 TDX-1A보다 용량을 2배나 증대시켰으며, 32비트 마이크로프로세서를 내장하고 있다.

이번에 설치된 곳은 경기도 안중전화국, 경북 경산·칠곡전화국, 강원도 주문진전화국 등이 있으며, 올해안으로 이외에도 전국 40개 전화국에 이를 설치할 예정이라 한다.

### –체신부
## 통신사업에 민간업체 참여를 유도

올하반기 한국전기통신공사 민영화에 따라, 체신부는 앞으로 전화사업에 있어 이를 시외와 국제전화사업으로 분리하여 경쟁체제를 유도함과 동시에 민간업체의 참여도 허용할 방침으로 있다.

이는 외국으로부터 가중되고 있는 통신시장 개방압력과 전기통신공사 자체의 비대화를 감안하여 전기통신공사를 민영화시키고, 현재의 시내·외 및 국제전화 사업을 독점하고 있는 체제를 개편하여 시내전화 부분만 현형태로 유지하고, 시외와 국제전화사업은 별도의 독립법인을 설치하여 민간업체의 참여를 유도하면서 경쟁력을 강화하는데 그 의의가 있다.

체신부의 기본구상은 시외전화의 경우 전국을 2-3개 구역으로 구분하고, 이를 각 구역별로 다시 별도의 법인형태로 운영하며 민간업체에게도 단계적으로 문호를 개방하여 경쟁체제를 유지한다는 계획이다.

이같은 체신부의 계획은 민영화된 선진국의 통신사업과 보조를 맞추는 좋은 일예이며, 국내 기업에게도 참여의 폭을 넓혀 기술개발을 꾀하는 좋은 기회가 될 것으로 본다.

–맥슨전자(주)
## 신용카드조회기 형식승인 취득

맥슨전자는 최근에 국내 처음으로 신용카드조회기에 대한 체신부의 형식승인(모델명 : AFI-8611K)을 취득했다. 이로써 맥슨전자는 그동안 제품마다 받아오던 사용승인 검사가 면제되어 생산효율을 높일 수 있게 됐다.

이번에 형식승인을 받은 단말기는 각종 조회 관련사항을 램프색으로 표시하여 이용편의를 한층 높인 것이 특징이다.

금번 형식승인으로 인해 맥슨전자는 올해 신용카드조회기 매출양이 지난해보다 50%가 늘어난 1만2천대선이 될 것으로 추정하고 있다.

–삼성 데이타시스템(주)
## 4세대 언어 NEW RAMIS외 5개 제품 국내 독점 공급

삼성데이타시스템(대표 : 임경훈)은 미국의 온라인 소프트웨어사와 실사용자 위주의 4세대 언어 및 DBMS를 결합한 소프트웨어 패키지에 대한 국내 독점 공급계약을 체결하였다.

뉴 라미스(NEW RAMIS)는 대형기종인 IBM 370, IBM 43**시리즈, IBM 30** 시리즈와 호환기종에서 사용할 수 있고 IBM PC 등의 PC급에서도 사용할 수 있다.

4세대 언어로서 모든 기능을 갖춘 뉴 라미스는 초보자도 쉽게 사용할 수 있도록 매뉴방식으로 운용되며 외부 데이터베이스와의 접속도 용이한 장점을 지니고 있다.

한편, 전산 전문가만이 다룰 수 있는 4세대 언어인 UFO는 신속한 응답시간, 기존 프로그램과의 호환성, 빠른 디버깅 특성을 갖고 있는 패키지로 자료사전을 보유하고 있으며, 현재 미국, 일본, 유럽 등 2500 SITI 이상에서 사용중이다.

이밖에 삼성데이타시스템은 개발 라이프 사이클을 총 관리하는 개발지원 툴인 INTELAGEN, 전자 우편시스템인 BULLTIN, DB2 개발지원 툴인 PROEDIT- DB2 제품에 대한 국내 독점 공급계약도 아울러 체결하고 소프트웨어 패키지 사업을 확대할 방침이다.

–금성반도체(주)
## 바코드판독 휴대용 컴퓨터 한글화 성공

판매관리시점 정보시스템(POS)과 함께 유통 및 공장자동화 핵심기기로 불리는 바코드 시스템에 사용되는 장비들이 한글화되어 유통 및 공장자동화에 활력이 되고 있다.

금성반도체(대표 : 이만용)는 최근에 상품의 제조일자, 생산지, 일련번호 등 상품의 고유정보를 선부호로 표시한 바코드를 스캐너를 통해 판독하고, 이를 데이터로 저장하는 휴대용 컴퓨터를 한글화하는데 성공했다고 밝혔다.

이로써 현장에서 제품을 취급하는 현장관리자는 누구든 손쉽게 이 휴대용 컴퓨터를 사용할 수 있어 유통 및 공장자동화에 크게 기여할 것으로 기대된다.

이번 한글화된 휴대용 컴퓨터는 기존 바코드 시스템에 접속하여 데이터를 처리하는 PC의 기능을 대신할 수 있는 장비로 신발, 의류 등의 매장 판매현황 파악 및 재고관리, 병원의 약품관리, 제약회사의 약품생산관리, 각종 제조업체에서의 창고관리 등에 폭넓게 사용할 수 있다는 장점을 지니고 있다. 또한, 이 휴대용 컴퓨터는 본체에 프린터를 부착하여 저장된 자료의 출력은 물론 간단한 영수증 처리도 가능할 뿐만 아니라 한글의 경우 8×4행자, 영문의 경우 16×4행자를 처리할 수 있어 이미 기록된 데이터와 현재 기록될 데이터를 비교하기가 용이하고, 전원이 끊어져도 수은건전지가 내장돼 데이터를 안전하게 보존할 수 있다.

자체진단 기능으로는 키버튼의 이상유무를 체크하여 바코드를 잘못 인식했을 때 신호음이 울리는 부저기능 등을 가지고 있어 보수유지 및 신뢰성을 높일 수 있다.

–금성 소프트웨어(주)
## 공장자동화 S/W CIM 보급

금성소프트웨어(대표 : 황칠봉)는 10일, 세계적 정보통신 전문회사인 미국의 LGS사와 최첨단 공장자동화 시스템인 통합생산관리시스템(CIM : Computer Integrated Manufacturing)을 국내 독점공급하기로 계약을 체결하여 이의 본격적인 공급에 나섰다.

이번에 금성소프트웨어가 공급하게 될 CIM은 종래의 부문별 공장자동화에서 탈피, 컴퓨터를 이용해 제품의 설계를 비롯한 생산계획, 설비제어, 작업관리 등 생산라인의 전 공정을 통합적으로 관리하는 효율적인 공장자동화시스템이다.

즉, CIM은 공장의 부문별 혹은 공정별로 설치되어 있는 기계 및 장비의 제어부에 퍼스널컴퓨터나 마이크로컴퓨터를 연결하여 최소의 인원으로 작업을 수행할 수 있도록 할 뿐만 아니라 이들 컴퓨터를 생산라인의 각 부문별 데이터를 통합조정하는 미니, 수퍼미니컴퓨터와 상호연결할 수 있게 구성됐으며, 각 공정별로 서로 필요한 자료나 명령을 신속하고 정확하게 주고받을 수 있게 했다.

또한, 이 미니나 수퍼미니컴퓨터는 본사의 호스트컴퓨터와 연결하여 공장생산라인에서 발생하는 모든 데이터를 경영 분석자료로도 사용할 수 있게 함으로써 제품생산에서부터 판매에 이르는 전 과정을 효과적으로 관리할 수 있다.

금성소프트웨어는 원가절감 및 품질향상에 크게 기여할 CIM이 많은 자본과 첨단기술이 요구되는 방대한 시스템인 점을 감안하여 국내 독점공급과는 별도로 이 시스템을 올해부터 연차

적으로 국산화에 착수하여 독자적인 공급체제를 갖출 예정이다.

### –금성통신(주)
## 호주에 키폰시스템 수출

금성통신(대표 : 임종염)은 호주 엑시컴사와 앞으로 4년간 2천만 달러 상당의 키폰시스템 제조기술 및 제품수출 계약을 체결했다고 지난 5월 3일 밝혔다.

이에 따라 금성통신은 올해안에 4백만 달러의 키폰 제조기술과 제품수출을 계획하고 있는데, 이번 키폰 호주수출은 최근들어 주력시장인 미국의 한국산 키폰 수입규제 움직임과 관련해 수출선 다변화 움직임으로 보인다.

동사가 수출하게 될 키폰은 자체개발품으로 중용량으로부터 최대 2백회선의 대용량에 이르는 4개 모델로 성능 및 디자인이 뛰어나 용량증설이 손쉬운 제품이다.

특히, 호주가 오는 7월부터 통신기기 수입을 자유화함에 따라 금성통신은 이번 키폰수출을 계기로 앞으로는 셀룰러폰, 코드리스트폰 등 무선전화기와 자동응답전화기 등 각종 다기능 전화기의 수출에도 본격적으로 나설 계획이다.

한편, 금성통신과 계약을 체결한 엑시컴사는 호주 최대의 통신기기 제조 및 판매업체로 미국, EC의 통신수출업체로도 널리 알려진 대기업이다.

### –롯데 파이오니아(주)
## 종합전자메이커로 탈바꿈 추진

롯데파이오니아(대표 : 신상호)가 산업전자분야 진출을 통해 종합전자메이커로 탈바꿈하고 있어 국내 전자산업의 판도변화가 예상되고 있다.

동사는 전자산업을 적극 육성한다는 그룹방침에 따라 하이파이 전문업체에서 탈피하여 컴퓨터, 반도체, FA, HA 등 고부가가치 품목으로 사업 다각화를 꾀하고 있다.

이에 따라 롯데는 종합전자사업을 시작할 경우 가전 3사와 현대에 이어 종합 5위권의 전자회사로 부상하기 위해 연구소내에 컴퓨터, 반도체, FA 등 신규참여분야의 제품개발에 필요한 연구팀을 신설하는 한편 개발인원을 계속 확충하고 있다.

컴퓨터의 경우 제품개발을 거의 완료한 상태이며, 10월경 본격 출하할 것으로 알려지고 있는데, 이때 선보일 제품은 1백만원대의 8088 CPU를 탑재한 16비트 XT와 2백만원대의 AT, 2기종이다.

또한, 내년도에는 80386 CPU를 채용한 32비트 컴퓨터 개발을 완료하는 한편 모니터 등 주변기기 개발에도 참여할 것으로 알려졌다.

반도체 분야는 하이브리드 IC를 우선 개발한 후에 타 반도체분야에도 확대할 방침으로 연구에 몰두하고 있다.

한편 FA분야는 롯데기공과 협력관계를 유지하면서 시장참여를 모색하고 있는데, 자동으로 전자부품을 조립하는 산업용 로보트 5개 모델을 개발하고 있어 본격적인 시장참여는 내년 하반기에 이뤄질 전망이다.

### –삼보컴퓨터
## 전국 대리점에 교육장 개장

삼보컴퓨터(대표 : 김종길)가 컴퓨터 업체로는 처음으로 전국 40여개 대리점에 교육장 시설을 갖췄다.

이에 따라 앞으로 트라이젬 유저라면 누구나 원하는 시간에 가까운 교육장에서 교육을 받을 수 있으며 또 제품구입과 동시에 교육을 받게되어 그 동안 신청 후 한두달 기다리던 불편이 해소되게 됐다.

삼보컴퓨터는 이와함께 전국 대리점 교육장에서 일주일단위로 MS-DOS, 보석글, 스프레드시트, 데이터베이스 Ⅲ 등의 교육을 무료로 실시한다고 밝혔다.

한편 삼보컴퓨터는 이번 전국 대리점에의 교육장 개설을 기념하기 위해 이달말 기획판매를 실시, 기간중 시스템을 구입하는 유저에게는 기본 유저 교육의 혜택과 GW-BASIC 메뉴얼 등을 무료로 증정하고 있다.

### –(주)쌍용컴퓨터
## 광주 창업상호신용금고에 SAMS 공급

쌍용컴퓨터(대표 : 최병항)는 최근 광주 창업상호신용금고와 자체개발한 신용금고종합관리 시스템인 SAMS 공급계약을 체결했다.

이로써, 창업상호신용금고는 여수신관리와 같은 창구업무로부터 고객관리, 인사급여관리, 경영정보관리와 같은 일반관리업무에 이르기까지 개발업무를 컴퓨터로 처리할 수 있게 됐다.

쌍용의 SAMS는 자금관리, 이익률분석, 영업속도, 통계자료 산출 등과 같은 각종 경영정보의 작성, 전달기능이 뛰어난 것이 특징이다.

현재 국내 상호신용금고업계는 총수신고가 5조원을 넘으면서 제1금융계와 금리차가 좁혀지고 공신력이 향상되면서 이용객이 부쩍 늘어 고객에 대한 신속하고 다양한 서비스제공과 대외 경쟁력 강화의 필요성이 부각돼 전산화가 활발히 추진되고 있는 실정이다.

쌍용은 이번 계약으로 전국 15개 신용금고사에 SAMS를 공급하게 됐다.

### –대우통신(주)
## COMDEX/Spring '89 컴퓨터 전시회 참가

대우통신(대표 : 박성규)는 인터페이스 그룹 주최로 지난 4월 10일부터 13일까지 미국 시카고에서 열린 COMDEX/Spring '89 컴퓨터 전시회에 참가했다.

COMDEX 전시회는 미국, 유럽, 아




시아 등지의 유수한 컴퓨터 전문업체들이 참가할 뿐만 아니라 특히 그 개최 시기상의 특징으로 하여 그 해의 신제품 출시현황과 개발동향 및 다음해의 신기술 추세를 점칠 수 있는 유명한 전시회이다. 대우통신은 XT, AT, 386시스템, PS/2, 랩탑, 프린터, 터미널, 팩시밀리, UPS(무정전 전원공급기) 등 총5개 품목 21개 기종을 출품하였고, 특히 386/2S 시스템 및 랩탑 PC 3개가 많은 관심을 끌었다.

한편 랩탑 PC는 슬라이딩 키보드 타입의 독특한 디자인으로 제작돼 눈길을 끌었는데 대우통신은 이 제품들에 대해 미국에 특허를 출원중에 있다.

뿐만 아니라 대우통신은 이번 전시회의 참가로 인해 자체 브랜드의 이미지를 한층 고양시킬 수가 있었으며 현지 언론들로부터도 좋은 반응을 얻어, 제2의 모델-D 신화를 창조하는 기폭제가 될 것으로 기대되고 있다.

**-(주)쌍용컴퓨터**

## 한글 전자사서함 기능 강화

쌍용컴퓨터(대표 : 최병항)는 최근 문서, 메모 등과 같은 각종 메시지를 컴퓨터로 작성하여 원하는 목적지와 송수신할 수 있는 한글전자사서함인 'CEMESS-11'의 기능을 대폭 강화한 새로운 시스템을 개발하는데 성공했다.

이 시스템은 8개월간에 걸쳐 개발을 완성했으며, 기존의 대화식 메시지 송수신기능과 자체에 내장된 편집기능외에도 각종 공고내용을 컴퓨터에 입력하여, 사용자는 언제든지 열람할 수 있도록 해주는 '게시판 기능'과 수신된 모든 문서를 종류별로 분류하여 알려주는 기능 등이 추가되었다.

특히, 게시판 기능은 각종 공고를 내용별로 분류하는 기능과 관리부서가 각종 공고의 게시여부를 승인, 점검하는 기능도 내장하고 있다. 또 기존의 시스템으로는 일반 사용자가 모든 부서와 메시지 송수신 현황을 열람할 수 있도록 되어 있어 시스템 반응시간이 매우 늦은데 비해, 새로운 시스템에서는 현황을 관리부서에서만 볼 수 있도록 기능을 보완하고 프로그램 논리 개선을 통해 시스템의 반응시간을 줄였다.

그러나 시스템 관리자는 모든 메시지의 송수신 현황을 열람할 수 있으며 초보자를 위한 도움말 제공기능을 보다 알기쉽게 보완하였고, 프로그램 규모도 축소되어 PC의 메모리를 최소화한 것으로 알려졌다.

쌍용컴퓨터의 한글전자사서함은 지난 '87년에 최초로 개발한 것으로 PC를 이용하여 각종 문서를 책상에서 작성하여, 곧바로 목적지와의 송수신은 물론 자동 결재할 수 있는 시스템으로서 기존의 정보전달 수단인 전화, 우편, 팩시밀리의 단점을 보완한 시스템이다.

또한, 이 시스템을 이용하여 문서의 유동이 수초만에 이뤄져 업무능률의 향상은 물론 경비절감 효과를 거둘 수 있어 향후 사무자동화를 위한 시스템으로 널리 보급될 것으로 보인다.

**-(주)원월드 코리아**

## 통신용 S/W QUICK-TALK 개발

원월드 코리아(대표 : 강상완)에서는 장거리 통신에서 초단위로 계산되어지는 현 전화국 서비스에 부응하는 진보 향상된 QUICK-TALK 소프트웨어를 개발하였다.

전화선 연결시간을 단축시킨 이 QUICK- TALK 프로그램은 전이중방식으로 양쪽 단말기에서 상호 동시에 메시지를 교환하게 된다. 2400 보오의 모뎀인 경우 1000자를 10초(실제 연결시간은 5초) 이내에 보내고 같은 시간에 상대방의 메시지도 받을 수 있어 통신비용을 절반 이하로 줄일 수 있을 뿐만 아니라 사용법 또한 간편하다는 특징을 가지고 있다.

소스코드는 C언어로 작성되어 있으며 (등록번호 89-01-24-0367) 판매가격은 12만원이다.
(문의전화 : (02)754-7777)

**-삼성전자**

## 9회 S/W 공모전 시상

삼성전자(대표 : 강진구)는 제9회 삼성S/W 공모전 시상식을 지난 4월 25일에 실시했다.

삼성전자는 일반에게 컴퓨터에 대한 이해를 돕고 퍼스컴의 보급을 확대하기 위해 일반 유저를 대상으로 매년 실시하는 이 공모전은 올해가 9번째로 1, 2월 두달간에 출품된 2백36본의 소

〔표 1〕 시상작 내용

| 구분 | 작 품 명 | 내 용 |
|---|---|---|
| 최우수 | 한국지명공부 | 각 도별 지도와 도시의 위치 및 특징을 알 수 있는 교육용 S/W |
| 우 수 | 성적처리 | 중학교 대상용 성적처리 S/W |
| | SLIP OTHELLO | IQ게임. 흰색과 검은색 돌을 이동하여 한가지색으로 통일 |
| | AUTO SOLVER | 마우스를 이용하여 그린 숫자와 수학기호들을 이해하여 계산해주는 패턴인식 S/W |
| 가 작 | SEG-1 | 간단하게 그림이나 도형을 그릴 수 있는 S/W |
| | CAI 물리 | 교육용 S/W |
| | MUSIC TOOL | 화면에 나오는 피아노 건반을 이용하여 피아노치기 작곡하는 S/W |
| | Mr Charles Chaplin | 게임 S/W |
| | 장기 | 2명이 장기를 둘 수 있는 게임 S/W |

프트웨어를 엄선하여 발표했다(표 1 참조).

이번 공모전의 최우수상은 우리나라의 각 도별 지도와 도시의 위치 및 특징을 알 수 있는 "한국지명공부"를 출품한 최원용(인천 남중학교 3학년)군이 차지했으며, 이외에도 모두 74본의 우수 소프트웨어가 상을 받았다.

상금으로는 최우수상에 1백만원 등 모두 1천여만원이 수여되었으며, 응모자 전원에게도 기념품을 증정했다.

특히, 이번 공모전 출품작중 초·중·고교생들의 작품이 전체의 55.2%를 차지해 컴퓨터에 대한 학생들의 관심과 활용이 매우 높은 것으로 나타났고 지역별로는 서울이 50%, 지방 6대 도시가 24.4%를 차지함으로써 퍼스컴 보급이 아직은 대도시에 편중되어 있는 것으로 나타났다.

## –삼보컴퓨터
## 한글 문서인식 시스템 발표

(주)삼보컴퓨터(대표 : 김종길)와 한국과학기술원 인공지능연구실 김진형박사팀이 공동으로 연구개발한 '한글 문서인식 시스템에 관한 연구' 발표가 지난 5월 19일 있었다.

실눈(SILNOON)이라 명명된 이번 연구발표는 과기처의 기업주도형 특정연구과제로 채택되어 지난 2년간 수행된 것으로 스캐너라는 시각장치를 통해 문서의 영상을 입력받아 문자영상을 추출한 후, 그 문자를 인식하여 컴퓨터가 사용하는 내부코드로 바꾸어 화일로 전송하거나 읽어주는 한글 문서인식 시스템이다.

실눈시스템에 사용된 시각장치로는 300DPI의 해상도를 갖는 스캐너를 사용했으며, 컴퓨터 기종은 트라이젬 386을 사용했다.

이번에 발표된 연구결과에서 문자 인식률은 99.5%에 이르고 있으며, 속도는 H4사이즈(레이저 프린터내의 글자 사이즈중의 하나로 신문글자보다 약간 크다.)의 용지를 스캐너로 읽어서 인식하는데 약 1분30초 정도 걸리는 것으로 알려졌다.

특히, 속도면에서는 다른 특별한 프로세서를 사용하지 않고 PC만을 사용한 것으로 한글문서 자동인식 시스템의 실용화가 가능하다는 것을 보여 줬다.

이번 발표에서 드러난 문제점으로는
- 인식 대상문서의 형태에 관한 제약 (신문의 경우에는 마우스를 이용하여 특정부분만 읽을 수 있다.)
- 문자의 크기와 활자체에 대한 제약 (보통 초당 5자 정도 읽고 있으며, 영문과 한글이 뒤섞여 있다든지 하면 이를 인식하는데 시간이 든다는 점)
- 종이질에 따라 영향 받음.

등의 음절단위로 인식하는 등의 한계가 있다.

## –캐드업계
## 국산화 1호인 「캐드메이트」 외면해

제1호 국산 범용캐드인 캐드메이트가 푼돈 5만원에 관련 매뉴얼을 비롯해 소스코드까지 공개됐으나 CAD/CAM 업체들이 이를 활용치 않고 있어 캐드메이트를 개발한 한국과학기술원 기술진들이 실망을 금치 못하고 있다.

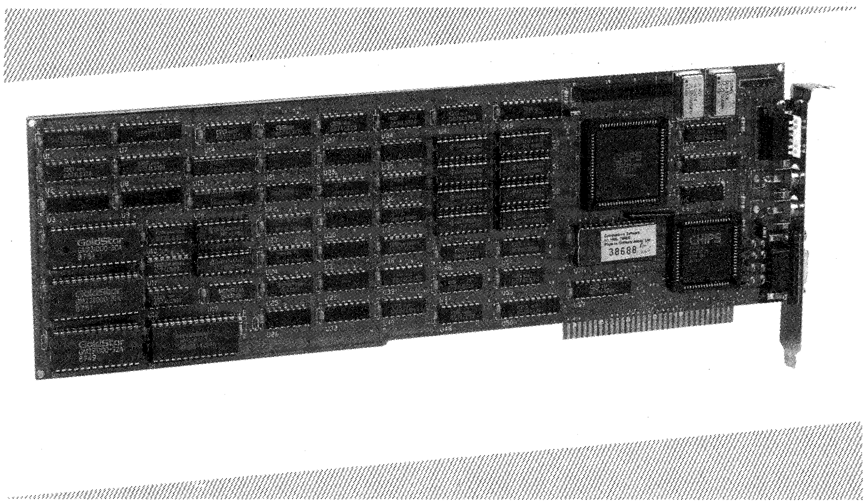
과기원 김기현박사팀이 지난 87년 과기처 특정과제로 총연구비 1억2천만원을 들여 캐드메이트를 개발하고 상품화작업을 벌이자 시기성 문제, 시장성, 사업성 문제들을 내세우며 관련 업체들은 이를 외면하는 대신 「캐드의 국산화를 위한 시금석」이라는 전제하에 캐드메이트의 원천기술 공개를 원했었다.

그러나 정작 캐드메이트가 공개되자 소스코드를 입수한 60여 관련 업체들이 이를 응용한 어플리케이션 개발은 뒷전으로 미루고 기존의 수입판매만을 고수, 당초 표명했던 국산화 의지는 단순한 호기심에 지나지 않았음을 입증하게 되었다.

이에 대해 과기원의 한 관계자는 「캐드메이트 개발팀이 갖가지 불이익을 감수하고도 국내 CAD/CAM 기술력 제고라는 차원에서 과감히 원천기술을 공개했음에도 불구, 소스코드를 얻기 위해 아우성쳤던 업체들이 이 기술을 사장화시키고 있는 것은 일종의 배신행위나 다를바 없으며 이는 향후 국내 CAD 기술확보에 찬물을 끼얹는 처사」라며 캐드 국산화의 어려움을 토로했다.

## –(주)옴니테크
## 중국어보드 수출

옴니테크(대표 : 최인철)는 중국시장 진출을 위해 우선 중국지역 유통망을 갖고 있는 홍콩의 올리베티사와 BIOS OEM 계약을 체결하여 5만 달러어치



의 중국어보드를 수출키로 했다.

옴미테크가 이번에 수출할 중국어보드는 퍼스컴에 부착하여 8천자에 달하는 중국어를 자유자재로 표시할 수 있을 뿐아니라, 중국 표준입력방식을 채택하여 사용하기가 쉽게 설계되어 있다. 또한 640×400의 고해상도 칼라 지원이 가능하며 다양하고 미려한 서체를 지원하고 있다.

동사는 중국어보드 수출이 본격화되는 올하반기부터 중국진출을 노리는 국내 퍼스텀 메이커들과도 OEM계약을 체결하여 보드 수출을 확대해가는 한편 중국에 합작 또는 단독출자를 통해 직접진출하는 방안도 신중히 검토하고 있다.

국내 퍼스컴 제조업체는 이번 옴니테크의 중국어보드 수출을 계기로 앞으로는 중국에 퍼스컴 수출을 더욱 촉진할 것으로 보인다.

–한국전자계산(주)

## 딜러에 리베이트제 도입

한국전자계산(대표 : 강원윤)은 PC CAD/CAM에 대한 유저들이 요구에 부응하고 기존 엔지니어링 워크스테이션의 성능제고를 위해 CV386 4개 모델에 대한 판매를 늘린다는 방침아래 최근에 딜러들의 경쟁력 강화를 통한 인스톨 베이스 확대에 적극 나섰다.

이를 위해 한국전자계산은 딜러에게 각종 인센티브 프로그램을 제공하고 딜러요원에 대한 해외교육을 실시했으며, 기존 유저들에 대한 무상공급 및 목표물량 초과분에 대해서는 2% 이상의 DC를 제공하는 등 각종 리베이트 제도를 도입했다.

또한, 한국전자계산은 신설 자회사인 한국캐드캠을 통해 올해말까지 CV386을 1백대이상 판매할 방침이다.

–현대전자

## 전자악기시장 참여

현대전자(대표 : 정몽헌)가 최근 세계적으로 수요가 늘고 있는 전자악기 분야에 본격 진출한다.

현대전자는 4월말부터 세계적인 전자악기메이커인 미국의 쿨즈웨일사와 기술제휴를 추진하여 5월말에 계약을 체결, 쿨즈웨일사로부터 기술지원을 받아 OEM방식으로 전자악기를 생산하는 한편 아시아지역 판매권을 획득하여 일본을 비롯한 동남아지역에 판매할 것으로 알려졌다.

이에 따라 국내 전자악기시장은 삼익 등 전문악기업체와 가전 3사, 인켈 등 기존업체와 치열한 경쟁이 예상된다.

이와함께 현대전자는 올하반기부터 쿨즈웨일사의 전자피아노인 250 시리즈를 국내에 판매하는 한편 자체적으로 이 회사 제품과 호환성이 있는 PC를 개발할 예정이다.

이번에 들어올 250시리즈는 매킨토시 PC와 연결하여 컴퓨터작곡이 가능한 최첨단 전자악기제품이다.

–알파컴퓨터 시스템

## 디지탈 테크놀러지 대리점 겸해

32비트 80386 PC, 3270 한글통신 에뮬레이터, 온라인 통신용 BSC카드, PC다중통신용 MUX, 병렬처리 컴퓨터, ROM 디스크 드라이브 등을 판매할 알파컴퓨터 시스템이 지난 5월 1일자로 개업하였다. 디지탈 테크놀러지의 경북지역 대리점을 겸하게 될 동사의 취급 품목은 다음과 같다.

- IBM COMPATIBLE PC386/AT/XT
- ROM DISK DRIVE
- 기타 PC 주변기기

주 소 : 대구시 달서구 두류3동 493-7
TEL : (053)60—3700

–(주)큐닉스

## AutoCAD용 프린터 드라이브 자체개발 성공

레이저빔 프린터와 도트 매트릭스 프린터를 생산하고 있는 큐닉스(대표 : 이대우)는 오토캐드의 출력을 자사의 프린터로 할 수 있는 프린터 드라이브를 개발했다.

현재 오토캐드내에는 레이저빔 프린터를 지원하기 위한 휴렛팩커드사의 레이저제트 플러스용 드라이버를 이용하고 있으나 이것은 이미지 상태로 처리하기 때문에 출력하는데 상당한 시간이 소요되고 있다.

이러한 단점을 보완하기 위해 큐닉스는 오토캐드의 제작회사인 오토데스크사의 ADI규격을 이용하여 레이저빔 프린터를 플로터처럼 사용할 수 있게 함으로써 출력속도를 향상시켰으며, 또한 시간과 비용을 절감시켰다.

# 서클동정

### ●인하대학교 컴퓨터써클●
## 세미나로 실력다져

인하대학교 컴퓨터써클(INCOM)은 지난 5월 2일부터 5일까지 인하대학교 5호관에서 춘계전시회를 개최하였다. 이번 전시회에서는 하드웨어로 사인보드와 소프트웨어로는 대리점 관리프로그램, 써클 운영전산화를 위한 써클 종합정보시스템, DTP를 이용한 신문발간 프로그램, 알파파테스트 등 10여 작품이 선보여 많은 호응을 얻었다.

동 써클은 지난 82년 3월 대학의 컴퓨터 연구개발 및 보급과 회원상호간의 친목도모 및 유대강화를 목적으로 창립되었다.

현재 회원은 정회원 121명, 준회원 169명, 명예회원 102명으로 구성되어 있고, 회원모집은 매학년 초에 컴퓨터 공개강의를 통하여 모집하고 있다. 신입회원들은 몇주간 실시되는 소정의 교육과정을 이수해야 하는데 이 과정에서의 테스트 결과와 출석율, 써클행사의 참가율을 바탕으로 정회원의 자격이 주어진다.

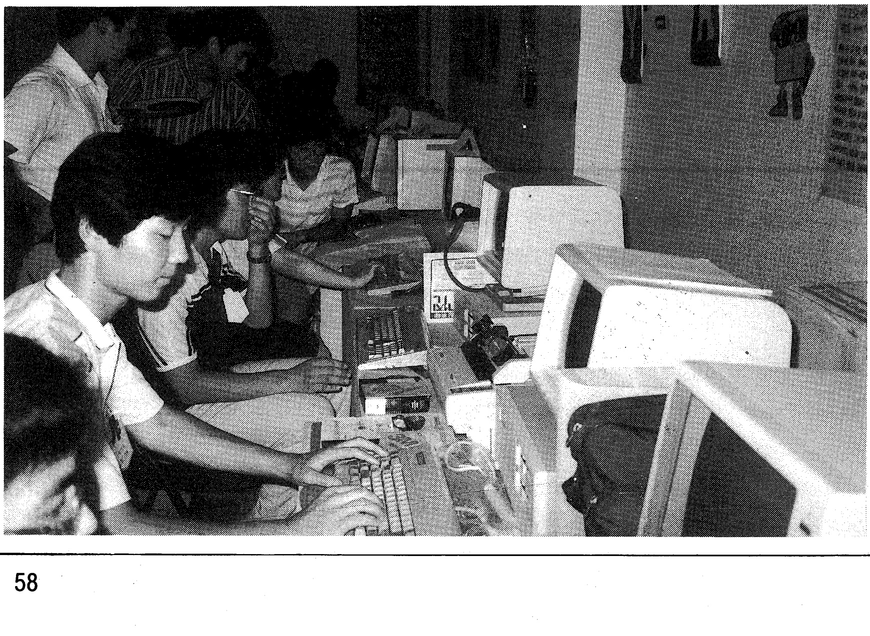
써클의 구성은 총무부, 학술부, 연구부, 편집부, 섭외부, 유니코사부로 구성되어 있다. 특히 연구부 산하에는 디지탈세미나팀, 제어회로 세미나팀, 응용회로 제작팀, Assembler 세미나팀을 두고 있고, 학술부 산하에는 워드프로세서 연구회, 데이터베이스 연구회, DTP 연구회, 그래픽 CAD 연구회와 각종 언어 세미나팀을 두어 각 팀마다 일주일에 3회씩 모임을 통하여 회원들을 교육하고 있으며, 매달 Compiler라는 학술문예지와 매년 "7~008"이라는 회지를 발간하는 등 활발한 활동을 펴고 있다.

또한 매년 겨울방학을 이용하여 인천지역의 고등학교 학생을 대상으로 무료컴퓨터 공개강의도 실시하고 있다.

이우섭 회장(전자과 3)은 이번 전시회를 계기로 회원들의 참여의식을 높이기 위해 회원교육을 강화할 것이며 가을에 개최될 프로그램 공모전시회의 준비를 위해 써클 자체의 실력향상 및 타대학의 활발한 참가를 위해 노력할 것이라고 한다.

*동 써클에 대한 문의사항은 전화번호 (02)567-0718 이우섭회장 앞으로 하면 된다.

DTP를 이용한 신문발간 프로그램이 가장 많은 인기를 끌었다.



### ●로터스 사용자 모임●
## 회원지 발간 다양화 예정

지난 2월 18일 로터스 사용시 문제점에 대한 상호정보교환 및 관련 유틸리티에 관한 연구를 목적으로 로터스 사용자 모임이 결성되었다.

동 모임은 로터스에 비교적 사용경험이 많은 회원 15명으로 구성된 A반과 로터스 사용의 초보단계의 회원 25명으로 구성된 B반으로 나뉘어 있고 각반의 회장에 의해 운영되고 있다. 회원에 대한 자격은 특별한 제한이 없고 회원을 희망하는 사람은 입회비 1만원과 회원 가입신청서를 제출하면 되고 회원이 되면 월 1천원의 회비를 내면 된다.

동모임은 매월 두번째 토요일 오후 3시 경희대학교 정경대 303호 강의실

에서 각반별 모임을 갖고 오후 4시에는 전체 모임을 갖고 있으며, A반은 매달 로터스에 관련된 주제를 정하여 발표회를 갖고 토론하는 형식으로, B반은 그룹별로 나뉘어 스터디를 통하여 로터스 사용에 관한 교육을 실시하고 있다.

또한 정기적인 모임의 계획과 회원간의 정보교환을 위해 매 모임마다 NEWS-LETTER를 발간하고 있다.

안병국 A반 회장(퍼킨엘머 코리아 관리부 차장)은 앞으로 NEWS-LETTER의 편집을 강화하여 소식 뿐만 아니라 로터스 관한 글과 더많은 로터스 관련 유틸리티의 활용을 소개할 것이며, HAL, Sideway 등의 한글 매뉴얼을 작성할 계획이라고 한다.

동모임에 관한 문의사항은 전화번호 (02)557-3571 안병국씨 앞으로 하면 된다.

●한라사랑방●

## 제주지역 정보문화확산 기수담당

지난 86년 8월 제주지역의 정보문화의 확산을 위해 제주지역의 각급학교 교사, 전산 전문인, 대학교수, 학생 및 일반인들 모임을 결성하였다.

회원은 한글전자 사서함의 가입자인 정회원과 가입예정자로 정회원을 통해 정보를 전달받는 준회원으로 구성되어 있고 현재 정회원으로 34개 가입자가 활동하고 있다. 회원에 대한 특별한 제한은 없고 가입을 희망한다면 한글전자사서함에 가입하고 전자사서함을 통해 가입의사를 밝히면 된다.

동 사랑방은 한글전자 사서함의 이용방법과 퍼스널컴퓨터 이용에 관한 교육 및 정보교환을 격월제의 모임을 통해 실시하고 있으며, 임원진으로는 고문, 지도위원, 회장, 간사 등이 구성되어 사랑방을 운영하고 있다.

이호식 회장(제주 그랜드호텔 전산실장)은 앞으로 전국적인 사랑방모임과 교류를 더욱 확대시켜 제주지역의 관광문화의 홍보를 담당하기 위해 노력할 것이며 컴퓨터와 통신에 관한 무료강좌의 개설과 정보교육개관이 주최하는 공개 세미나 등을 주최하여 일반인들이 쉽게 컴퓨터와 접할 수 있도록 교육을 실시할 것이라고 한다.

동사랑방에 관한 문의 사항은 전화번호 (062)42-3321 (교환 227, 228) 이호식씨 앞으로 하면 된다.

●토피아 유저클럽●

## 같은 수준에 따라 팀을 구성

작년 10월에 창립된 토피아 유저클럽은 고객에 대한 서비스제공이란 차원에서 시작되어 현재 150여명에 이르는 회원을 확보하고 있다.

컴퓨터마인드 확산과 컴퓨터 이용의 극대화를 위해 결성된 만큼 회원의 구성도 다양하게 구성되어 중고생, 대학생, 일반인 등 여러 계층에서 참여하고 있다. 또한 회원들을 컴퓨터의 이용정도로 구분하여 팀별로 구성하여 한팀당 10명 정도로 짜여져 상호간의 정보교환 및 교육을 같이 받고 있다. 지금은 베이직언어의 응용과 업무용 소프트웨어의 개발능력 배양에 중점을 두고 있으며 아울러 BBS의 도입을 진행하고 있는데 BBS가 도입되면 회원간의 교류가 더욱 활발해 질 전망이다.

회원들이 모임을 자주 가질 수 있도록 교육장을 개방하여 활성화를 꾀하고 있지만 아직은 걸음마 단계로 미비한 점이 많은 편이다. 현재 회원에 가입하면 컴퓨터에 기초적인 내용을 교육받고 팀에 합류하게 된다. 한팀은 약 10명 정도로 구성되며 컴퓨터이용 수준에 따라 분류되는데 현재는 6개의 단계로 나누어져 운용되고 있다.

클럽에 가입을 원하는 사람은 PC를 보유하고 있는 사람으로써 직접방문하여 신청서를 제출하면 된다. 회비는 무료이고 연락은 이동명씨(전화 : (02)544-1468)에게 하면 된다.

●두산 오리콤●

## 두산그룹내 최초의 순수 유저서클

두산 오리콤에 PC유저클럽이 결성되어 사내에 관심이 고조되고 있다. 지난 2월에 결성된 이 모임은 아직은 초보적인 단계에 머무르고 있으나 PC의 보급이 잘되어 있는 오리콤회사의 특성에 비춰 볼때 이 PC동호회는 무한한 발전가능성이 잠재되어 있다고 할 수 있다. 현재 주 1회의 모임으로 학습위주로 DBASE교육이 진행되고 있다.

회장을 맡고 있는 김철민(오리콤전산부)씨는 그동안 짧은 지식이나마 서로 협력하며 학습한 결과가 나타나고 있는데 대하여 보람을 느끼고 있다고 한다. 앞으로 현재 모임활동을 통하여 제작된 프로그램을 보강하고 개발하여 사내전시회를 개최하는 것이 1차적 목표라고 한다. 이 모임의 특징은 애플컴퓨터부터 16비트컴퓨터를 사용한 사용자와 처음으로 컴퓨터를 사용하는 사용자가 함께 어울려 진행되고 있는 순수하고 소박한 PC동호회이다.

회원의 조건으로는 오리콤에 근무하고 있는 직원으로 규정하고 월 3000원씩의 회비를 걷고 있는데 이 회비는 새로운 소프트웨어의 구입과 도서구입 등 주로 정보구입에 이용되고 있다. 이 모임에 가입하고자 하는 사람은 전화 (02)510-4185 강 태형씨에게 연락하면 된다. PC

# 국외소식

## IBM
## CIM 소프트웨어 발매

미국IBM은 CIM(컴퓨터에 의한 생산통합시스템)대응의 소프트웨어 2종류를 발표하였다. 이번에 발표된 소프트웨어는 엔지니어링작업을 지원하는 소프트웨어와 컴퓨터에 의한 설계통합을 위한 소프트웨어인데, IBM이 CIM 대응형 소프트웨어를 판매하는 것은 이번이 처음이다.

CIM은 부품, 자재의 조달 등 엔지니어링이나 제조, 판매, 유통의 생산에 관련하는 모든 단계를 컴퓨터에 의해서 유기적으로 콘트롤하는 구조이다. 이 CIM은 미국에서 조금 먼저 실용화 되었고, 일본에서도 실용화가 시작되고 있다.

소프트웨어의 가격은 중형기용에서 2종류 모두 라이센스료가 각 2천달러 정도이고, 올 7월부터 발매할 예정이다.

## 미, NCR
## 전제품 RISC 탑재 예정

미국의 대형 전산기메이커인 NCR이 컴퓨터 전제품에 고속계산을 가능하게 하는 RISC(Reduced Instruction Set Computer : 축소명령세트 컴퓨터) 칩을 장착할 것을 표명함으로써 이제까지 워크스테이션이나 미니컴과 같은 소형기의 전유물이었던 RISC 칩이 앞으로는 대형기에도 사용될 전망이다.

RISC는 컴퓨터의 기능이 다양해지면서 늘어난 명령어수로 인한 처리속도의 저하에 대한 대책안으로 마련된 것으로, 기능은 다양하지 않더라도 사용빈도가 높은 명령어들만을 모아 제어계통을 간소화하여 처리속도를 향상시킨 칩이다.

이러한 RISC칩은 IBM이 처리속도를 향상시키기 위한 방법으로서 미니컴인 801에 채용한 것이 시발점이다.

현재 이 기술을 MPU에 탑재한 것으로는 모토롤러 'M88000'과 선마이크로시스템의 'SPARC', 밉스컴퓨터의 R3000등이 있다.

미 NCR사는 우선 소형기내에 RISC 칩을 탑재할 계획임을 밝혔다. 따라서 그 1단계로 OS상에서 UNIX를 사용한 소형기인 'TOWER'를 내년에 발매할 계획이다. 'TOWER'는 CPU로 모토롤러사의 마이크로프로세서 M88000을 사용하는데 이 M88000의 현 처리속도는 수십 MIPS(1초간에 100만회의 명령을 실행)이다. 모토롤러에서 1991년까지 100MIPS의 고속타입을 개발할 계획이 진행되고 있다.

2번째 단계로 NCR은 TOWER에 이어서 NCR9800이나 NCR10000 등의 범용기종에도 RISC칩을 탑재할 계획이다.

이와 같은 NCR의 계획은 아직까지 대형 범용기 세계에서 압도적으로 시장을 점유하고 있는 IBM에 비해 고전을 겪고 있던 동사가 시스템의 대형화를 피하고, RISC를 채용하여 처리성능을 향상시킴으로서 시장점유율에서 어느 정도 회복될 것으로 예상된다.

이제까지 RISC 기술을 사용한 제품은 CAD/CAM이나 과학기술계산, 화상처리 등 연구·제조분야에서 많이 사용되어 왔다. 앞으로도 소형기의 경우는 문제가 없지만 소형보다 대형의 범용기에서 사용하려면 사무실에서 일어나는 다양한 수요에 응할 수 있도록 RISC기술을 보강할 필요가 있다.

## 미, 인스타트사
## 올 반도체산업 호황예측

금년 미국 반도체수요전망에 대해서 마이너스성장을 예측하고 있었던 시장조사회사인 인 스타트사는 최근들어 13% 정도 성장되리라고 예측을 "상향" 수정하였다.

동사의 젬 빅터사장은 "수주가 3월달 수준으로 진행하면 작년의 제4분기를 상회하는 새로운 기록을 올릴 공산도 강하다"고 말하였다.

인 스타트사는 작년말을 기점으로 금년의 미국반도체시장은 3.6%감소되리라 예측하였는데 그 이유는 'DRAM, EPROM, MPU(마이크로프로세서) 외의 제품 값 하락이 89년의 성장에

〔표 1〕

(금액 : 10억달러)

| | 1988년 | 1989년 | 증감(%) |
|---|---|---|---|
| 미반도체 산업전체 | $11.5 | $13.2 | 14.5 |
| (내부디스크리트 IC) | $ 1.91 | $ 1.90 | 3.7 |

(자료 : 인스타트사)

큰 영향을 미치게 될 것'이라는 동사의 지적이었다.

그러나 동사가 이렇게 예측을 수정한 것은

① 제조가 확대되었다.
② 컴퓨터산업이 예상대로 호황국면이다.
③ 엔드 유저재고가 전반적으로 조정되었다.

등에 의한다.

동사의 수정예측에 의하면 금년의 미국반도체시장은 전년대비 14.5%증가한 132억 달러로 성장할 것이고, 이중 디스크리트 IC시장은 3.7%증가한 19억 8천만 달러에 달할 전망이다. (표 1 참조).

일, TDK

## 반도체메이커인 「실리콘시스템」사 매수키로

일본 TDK는 미국의 유력한 반도체메이커인 실리콘 시스템즈(본사 캘리포니아주 다스틴)의 주식을 공개매입(TOB)하기로 동사와 합의하였다. 이대로 매수가 실현되면 TDK는 매수비용으로 2억달러를 상환하여 반도체사업에의 진출을 달성할 것이다. 그러나 TDK가 실리콘 시스템즈를 매수함으로서 동사에 돌아갈 이익이 어떤 것인지에 대한 견해는 분분하다.

전자부품은 전자기기, 가전제품 등의 고기능화, 소형경량화의 흐름에 따라서 될수록 많은 기능을 작은 부분에 수용하기 위해서 성능개선과 소형·박형화의 양립을 강하게 요구하고 있다. 그러나 IC도 얼마만큼의 전자회로를 하나의 반도체칩에 집적할 수 있느냐를 본다면 이 또한 지향하는 바가 같다.

이런 점에서 전자부품과 IC는 서로 경쟁대상이 된다.

더구나 특정한 용도에 맞춰 종래의 전용IC보다 저가로 생산되는 ASIC(특정용도용 IC)이 앞으로는 더욱 보급될 추세인데 반해 일반 전자부품의 열세는 어쩔 수 없을 것 같다. 그러나 IC가 모든 전자부품을 대신할 가능성이 있다고 하여도 그렇게 되기까지는 상당한 시일이 걸릴 것이다. 그러나 이러한 현상은 전자부품메이커로서는 심각한 문제이다.

그러나 전자부품과 IC를 모두 취급하는 업체는 기존시장을 유지해가는데 크게 염려할 필요가 없을 뿐 아니라 오히려 양쪽을 조합한 신제품을 개발하여 새로운 새시장을 개척할 수 있다.

또 반도체제품에 사용되는 스파터링 기술 등은 TDK의 자기헤드나 센서의 기술혁신에도 많은 영향을 미칠 것이다.

전자부품의 납입선인 세트메이커의 동향이나 요구를 정확하고 빠르게 포착하기 위해서도 반도체산업과의 연계성은 반드시 필요하다. 따라서 철강메이커가 신규사업으로서 참가한다든지, 자동차메이커가 자사용으로 개별적인 IC를 내재화하는 경우와 비교해보면 TDK의 이러한 결정은 본사업의 방향과 강화에 보다 더 많은 비중을 차지한다고 볼 수 있다.

전자부품업계에서는 어느 정도 전자회로나 기구부품을 조립한 비완성품을 만드는 새트화의 흐름 역시 강해지고 있다. 이러한 현상을 조합해 보면 단지 IC를 구입하여 시스템만을 만드는 방법보다는 자사에서 개발·생산할 수 있는 쪽이 효과적이라는 해석이다.

더욱 핵심을 찌르는 견해도 있다. TDK 는 생각하고 있지 않다고 부정하고 있지만 가전, 전자기기 등의 시스템 사업에서의 포석이라는 해석이다. 동사는 다른 전자 부품메이커와 달리 테이프 등으로 소비자쪽의 지명도가 이미 높아, 전기제품의 판매루트도 만들기 쉽다.

경쟁하지 않는 신규 제품시장이면 시스템쪽으로 진출할 가능성은 있다고 볼 수 있다.

실리콘 시스템즈의 매출액 약 8할을 하드디스크 드라이브(HDD)용의 IC가 차지하고 있는 것은 HDD용 헤드를 취급하는 TDK와 전혀 연관이 없는 것은 아니다. 단, HDD 완제품을 시작할 가능성은 HDD 메이커와의 관계를 맺으면서 이루어질 것이다.

TDK가 일본의 IC메이커를 매수하지 않더라도 기술적인 측면만이라도 일본의 IC메이커와 손을 잡아도 좋았을 것을 굳이 미국메이커를 매수할 필요가 있었을까 하는 의문이 일지도 모른다. 그러나 일본의 반도체메이커는 대부분 완제품 메이커이며, TDK의 거래처이다. 따라서 특정한 반도체메이커와 손을 잡는 것은 특정한 유저와 손을 잡는 것이 되기 때문에 부품메이커로서는 일을 하기 어렵게 된다. 이러한 일본특유의 반도체업계의 구조로 인하여 TDK는 국외기업을 매수하든가, 모든 일을 혼자힘으로 해 나가는 수 밖에 없었을 것이고, 혼자 힘으로는 시간이 너무 걸린다는 결론에 도달하여 결국에는 국외기업을 매수하기로 결정하였을 것이다.

## 미 반도체산업
# 고속SRAM 시장, 성장과 혼전예상돼

요즈음 미국에서는 PC(퍼스컴)와 WS(워크스테이션)시장이 활발해지면서 고속 SRAM(35~70ns)시장이 급성장하고 있다. 사실 이 SRAM시장은 9억5천만달러(1989년)에서 10억달러(1990년)로 약간 증가한 후 2년 뒤에는 20억달러(1992년)로 대폭 증가하는 높은 성장성이 예측되고 있다(그림 1). 반면 저속 SRAM(70ns이상) 시장은 8억2천5백만 달러(1988년)로 이미 최정상에 올랐으며 앞으로는 내리막 길로 접어들어 6억5천만달러(1992년)로 점차 약간씩 감소하여 가리라고 추정되고 있다.

고속 SRAM(35ns 이하) 시장이 급성장하게된 배경에는 2대 요인이 있다. 첫째는 1MB DRAM시장의 수급완화가 도래한 것, 둘째는 상위 PC와 WS용으로 32비트 RISC CPU 아키텍처가 잇달아 등장하고 있는 것 등이다. 이 시류에 뒤떨어지지 않으려고 미국과 일본등지의 10여사가 일제히 이 시장에 급히 참가하고 있다(표 2).

첫째 요인인 DRAM시장의 수급완화는 일본의 경우 대반도체메이커가 자사의 여유있는 생산라인을 갖고 있어 아직은 볼황상태에 있는 SRAM제조에 참여가 가능하기 때문이다. 원래 DRAM시장의 3분의 1규모라고 하는 SRAM시장이 혼전되는 것은 불가피한 일일 것이다.

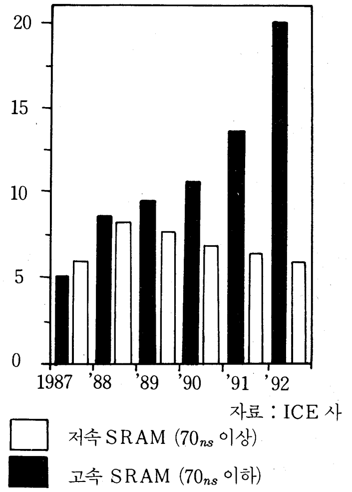
둘째 요인은 (초)고속SRAM이 컴퓨터시장의 주역인 32비트 PC/WS의 주요 보좌역의 위치를 차지하는 것을 의미하고 있다.

WS의 32비트 CPU가 RISC화되는 것이 앞으로 주류를 이룰 것은 틀림없다. 선 마이크로, MIPS에 이어서 인텔(i860/80486)과 모토롤러(88000/68040)도 RISC형 CPU군을 속속 등장시키고 있다. WS메이커들도 CPU로 RISC를 채용할 계획을 수행중이다.

앞으로 고속SRAM시장은 WS수요가 견인격으로 되어 확대해 가고 있고 (그림 2), SRAM메이커는 이 수요에 밀착한 신제품개발, 생산, 공급작전을 실행해 나갈 것이다. 그리고 이 전략은 가격의 유리성, 아키텍처의 점유화, 어플리케이션(고객)확보의 용이성 등의 이점과 우위성을 가져 오게 될 것이다.

고속SRAM에 대한 앞으로의 또 하나의 주류는 BI-CMOS화이다. 이 BI-CMOS는 바이폴러의 고속성과 CMOS의 저소비전력이라는 하이브리드 특성

〔그림 1〕 SRAM 세계시장규모의 1987년~1992년 추이 (판매전액/단위 달러)

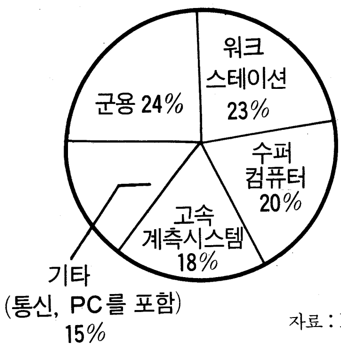



(주) 1989년 이후에는 추정한 것임

〔표 2〕 미국 고속SRAM시장 주요메이커

| 유형 | | 메이커군 | 소재지 | 현황 |
|---|---|---|---|---|
| 전문업체 | | Cypress Semiconductor | 산호세 | • 인텔(80386)계 CPU와 선마이크로시스템즈(SPARC)계 RISC CPU용 고속SRAM의 특별화중 |
| | | Integrated Device Technology (IDT) | 산타클라라 | • MIPS (R3000)계 RISC CPU용 고속 SRAM의 특별화중 |
| 신흥업체 | | Aspen Semiconductor | 산호세 | • 3ns의 초고속 SRAM 개발실현 |
| | | Synergy Semiconductor | 산타클라라 | |
| | | Paradign Technology | 산타클라라 | • 89년¼ 분기 초고속 SRAM개발 |
| 대형 | 미국 | AMD | 산타클라라 | • 1년전 1번 후퇴하고 고속SRAM시장으로 잠입. 소니에서 제조 프로세서 도입 |
| | | National Semiconductor | 산타클라라 | • 1년전부터 동사 Puyallup (워싱턴주) 공장에서 SRAM 생산개시 |
| | | TI | | • 군용 SRAM에서 범용고속SRAM 시장으로 이동중 |
| | 일본 | 히다찌, 후지쯔, 미쯔비시, 도시바 | | • 특정 RISC CPU 아키텍처용 개발<br>• 채산시장규모(10억달러)로 향한 양산체제 확립중 |

〔그림 2〕 1988년 고속SRAM의 주요 미국 수요분포




자료 : ICE사

을 살려서 현세대의 SRAM가격으로 다음 세대의 고속SRAM을 공급하는 것이 그 목표이다. 고속SRAM의 BI-CMOS화는 연 33%비율로 신장하여 그 시장규모는 앞으로 5년 동안에 4천만달러(1988년)에서 4억달러(1993년)로 10배 증가할 것으로 예상되고 있다.

그렇다고 해도 고속SRAM시장이 모두 장미빛이라는 것은 아니다. 도전할만한 과제도 있다. 예를 들면

① 최첨단의 고밀도·초고속 SRAM을 가능한 단기간내에 개발·생산하기 위한 독자적인 기술과 자사의 생산공장을 가지고 있는지의 여부,

② 유저의 빠르고 다양한 요구에 대응하여 다품종 소량생산체제가 이루어지는지의 여부

DRAM경쟁에서 일본에 완패한 미국 반도체메이커군은 고속SRAM으로 새국면에 돌입하면서 급성장이 약속되고 있다. 과연 이 고속 SRAM시장에서 그 교두보를 구축할 것인지가 주목된다.

### 컴퓨터 메이커
## 대부분의 퍼스컴에서 EISA 규격취득

컴팩사를 중심으로 하는 EISA(Extending Industry Standard Architecture)의 공동개발멤버 9사는 요즈음 EISA의 개발이 순조롭게 진행되고 있어 규격서의 주요부분은 이미 작성이 완료되었으며 지금까지 100을 넘는 메이커나 개발회사가 EISA제품을 개발하기 위해서 이 규격서를 취득하고 있다고 발표하였다.

250페이지가 넘는 EISA의 규격서에는 32비트 코넥터의 규격을 포함하고 있다. 이 코넥터에는 종래의 8, 16비트용 보드에서는 불가능했던 2중구조를 가능하게 하여 32비트의 어드레싱능력을 발휘하며 상하코넥터간의 인게이지먼트(engagement : 맞물림)를 완전하게 하기 위해서 칼자국이 들어 있다. 사이즈는 현재의 ISA(표준규격아키텍처)에 의한 16비트 코넥터와 같다. 이 32비트 코넥터의 설계는 번디사, AMP사 등 EISA 2 그룹 메이커 벤더의 공동개발에 의한 것이며, 양산은 2/4분기에 개시될 예정이다. 한편 EISA 규격 IC의 엔지니어링 샘플도 제 2/4 분기중에 인텔사에서 발표될 예정이며, 컴퓨터 메이커로의 출하는 금년 후반기로 예정하고 있다.

따라서 주변기기를 포함하는 EISA의 기본 퍼스컴 시스템의 판매개시는 금년말이 될 공산이 크다. 유저들 사이에서는 당초 25MHz의 80386을 사용한 고기능퍼스컴이나 다음 세대의 80486을 탑재한 머신 등 하이엔드의 제품에 사용되리라는 견해가 유력하다. 신제품의 가격에 대해서는 현재의 AT기종보다 높을 것은 확실하지만 IBM의 PS/2보다는 저렴할 것으로 보고 있다.

한편 EISA 그룹의 메이커들 중에서도 IBM의 MCA(Micro Channel Architecture) 기의 개발을 계획하고 있는 곳도 많다. 예를 들면 텐디사는 이미 MCA기를 판매중이고, 델 컴퓨터사의 마이켈 델회장은 "유저입장에서 관심이 있는 것은 CPU, 메모리, 디스크의 성능일 뿐 EISA와 MCA의 개발경쟁에는 그다지 흥미를 나타내고 있지 않다"고 말하고 있다.

### 미국 퍼스컴업계
## 새로운 유닉스머신 잇달아 발표

UNIX가 AT&T사 벨연구소에서 처음으로 인스톨되고 나서 금년으로 20주년을 맞는다. 그동안 LSI기술의 급속한 진보에 따라서 컴퓨터의 크기는 100분의 1 사이즈로 콤팩트해졌고 반면에 속도는 수100배에 달하고 있다. 또한 DEC사의 PDP-11에 비해 고속인 머신이 각 가정에까지 잠입하고 있다.

이러한 상황에서 UNIX쇼인「UniForum」도 금년으로 제6회째를 맞으면서 유래없는 대성황을 이루었다. 그 배경에는 80386계 머신상에 UNIX이식이 거의 완료되어 PC레벨의 머신에서도 UNIX가 사용 가능해졌다는 것과 고성능인 RISC칩의 등장으로 상위UNIX 머신의 성능이 비약적으로 증대하였다는 점들이다.

이들 2가지는 유저의 입장에서 볼 경우 UNIX머신의 선택폭이 확대되었다는 점과 그 결과로서 UNIX는 워크스테이션레벨의 머신을 중심으로 공통 OS로서의 강력한 지반을 확립하게 되어 그 의미는 크다하겠다.

이중에서 SPARC칩은 Sun-4 시리즈에서 사용되고 있는 RISC칩이며, 10~20MIPS의 처리속도를 자랑하는 CPU 이다. 그 아키텍처를 그대로 채용한 Sun-4 호환기종이 Solbourne사에서 발표되었다(동사는 마쯔시다전기와 제휴관계에 있다). 동사의 Allan Snell에 의하면 「Series 4」라 불리우는 이 워크스테이션은 Sun-4와 바이너리레벨에서 완전한 호환성을 가지고 있으므로 어플리케이션을 변경하는 일 없이 그대로 동작시킬 수 있다고 한다. PC

# 명령어의 활용과 응용설계

김용득/아주대 전자과 교수

지난호에서는 명령어를 구성하기 위한 기본원리를 설명하였다. 따라서 본 호에서는 실제적으로 사용되고 있는 명령어를 중요 그룹별로 나누어 예를 들어 설명하기로 한다.

8086에 사용되는 명령어는 약 300여개 정도가 있으나 이를 기능별로 구분하면 다음과 같다.

① 정보의 이동을 위한 명령어(Data Transfer)
② 연산처리를 위한 명령어(Arithmetic)
③ 비트처리를 위한 명령어(Bit Manipulation)
④ 문자처리를 위한 명령어(String Manipulation)
⑤ 명령어 처리순서 등의 제어를 위한 명령어(Control Transfer)
⑥ 프로세서 제어를 위한 명령어(Processor Control)

여기에서 볼 수 있듯이 비록 300여개의 명령어를 갖고는 있으나 모두 상이한 것이 아니고 유사한 종류 몇개로 구분되어 있다. 이외에도 고급명령(ENTER, LEAVE 등이 있으나 명령동작으로 분류하기 때문에 제어 전송명령에 포함됨)과 시스템 제어명령(메모리관리, 보호, 작업관리 등) 등이 있다.

독자에게는 보편적인 기호로 된 명령어(이를 어셈블리어라 부름)를 사용하여 프로그램을 숙지하길 권한다.

## 정보이동을 위한 명령어

먼저 정보의 이동을 위한 명령어를 설명하겠다. 정보의 이동을 위한 명령어는 그림 1과 같이 프로세서의 내부레지스터, 기억소자 및 입·출력에 연결된 포트 버퍼 간을 이동하며, DMA(추후에 설명됨)방식의 이동명령을 제외하고는 모두 프로세서 내부레지스터와 입·출력장치, 주기억장치 및 다른 프로세서의 내부레지스터에 저장된 정보간을 이동한다. 물론, 이때 이동되는 위치 및 정보의 형태에 따라 지난호에서 설명한 주소지정방식이 그대로 적용된다.

입·출력장치와의 정보이동을 제외하고는 대부분 "MOV dst, src"로 이루어졌다. 여기서 dst는 정보가 이동되어 기억될 위치이며, src는 정보가 이동되기 전에 기억되어 있던 근원지를 나타낸다. dst 및 src는 마이크로프로세서의 내부레지스터이거나 기억소자의 위치번지, 또는 정보값이 될 수도 있다.

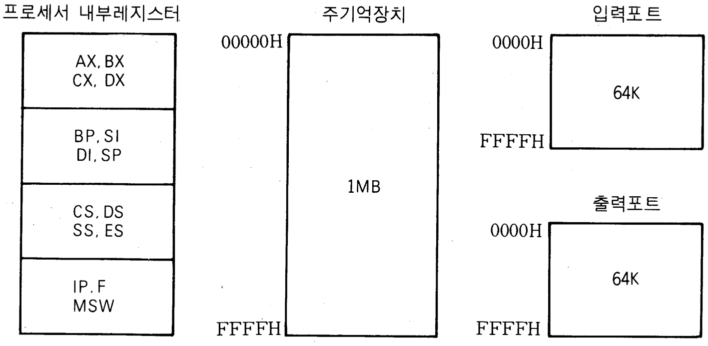
따라서 MOV 명령어는 이들 적용형태에 따라 28개의 서로 다른 기계어 구성형식을 갖게 된다.

외부의 입·출력장치와 정보가 이동되는 방식은 기억소자를 사용한 입·출력방식(Memory Mapped I/O)과 포트방식을 사용한 입·출력방식(Port I/O 또는 Isolated I/O)이 있으나 8086에서는 주로 후자의 방식이 사용된다.

이 포트방식의 입·출력은 IN과 OUT이라는 명령어로 처리되는데 8비트 또는 16비트에 따라 정보가 이동되는 내부레지스터로 각각 AL, AX가 사용된다.

물론, 포트 지정방식에서도 직접 주소지정방식과 간접 주소지정방식으로 하드웨어를 구성할 수 있는데, 직접지정방식은 256개의 입·출력장치를 직접 명령어에 지정하는 방식으로 8086에서는 주로 이 모드를 사용하고 있다. 반면에 간접지정방식은 64K개의 입·출력장치를 16비트로 지정하는 방법으로서 프로세서의 내부레지스터인 DX

〔그림 1〕 정보이동을 위한 프로세서 시스템영역



를 통하여 입·출력장치를 지정하는 모드를 가리킨다. 즉, DX레지스터에 입·출력장치의 주소를 지정하고 AL (또는 AX)를 사용하여 정보를 입·출력한다.

이들 입·출력장치를 지정하기 위해 **그림 2**와 같이 입·출력 주소지정표를 작성해야 하며, 지정번지(이를 포트라 부름)에 특정 입·출력장치를 할당시켜야 한다.

입·출력장치를 컴퓨터 시스템에 접속하기 위한 하드웨어 회로와 운영 작동방안을 예로 들어 좀더 자세히 설명하겠다.

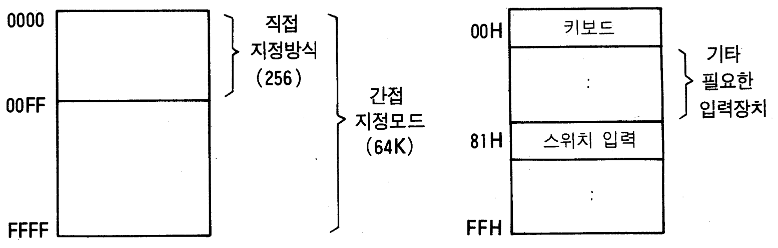
먼저 입·출력장치에 관한 하드웨어적인 접속회로는 모두 **그림 3**과 같은 블록으로 도시된다. 입력장치로부터의 정보신호는 정보값과 유효한 정보값임을 알려주는 제어신호로 구성되며, 이 제어신호에는 포트 접속회로의 래치에 입력장치로부터의 정보를 기억시키는데 사용된다. 그러면 마이크로프로세서는 접속회로의 버퍼소자에 새로운 정보가 있는가를 검사(검사방법은 핸드세이킹 모드, 인터럽트 모드에 의하여 이루어지며 다음호에 이를 설명하도록 한다)하여 입력 접속회로를 구동하기 위해 요구되는 신호를 발생하므로 이를 입력하면 된다.

**그림 4**에는 실제 키 입력장치로 사용될 수 있는 접속회로를 보였다. 여기서 SN74LS573이라는 접속소자는 그림 3의 8비트 래치회로와 3-상태 버퍼로 구성된 것으로 입력장치와 프로세서 시스템 사이의 작동시간에 대한 동기를 일치하는데 사용된다.

여기서 장치지정 논리회로는 일반적으로 디코더 회로가 사용되나 간편하게 하기 위해 NAND 게이트로 구성하였다. 이때 게이트 입력은 입력지정 포트구성에 따라 설계자가 임의로 지정하게 되는데 본 예에서는 그림 4(b)와 같이 입·출력장치를 할당하였다.

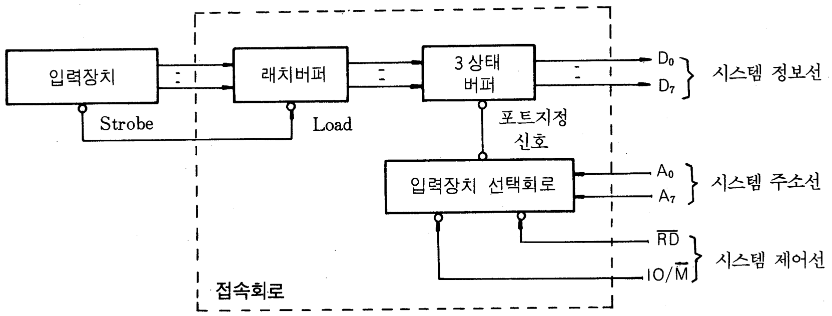
이들 회로의 작동은 프로그램상에서 "IN AL, 81H"를 수행하므로써 처리되는데, 이때 제어신호는 내부동작에 일치되어 $\overline{RD}$, M/$\overline{IO}$신호가 지정된 주소번지와 함께 프로세서로부터 출력되며, 지정 입력장치(본 예에서는 81H 번지)로부터 8비트 정보를 읽어 AL에 기억한다.

〔그림 2〕 입·출력 주소지정표

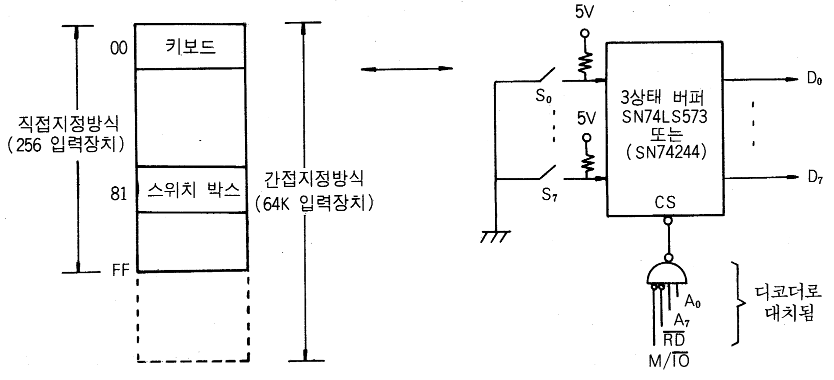



(a) 입·출력 지정모드 (b) 입력장치의 지정 예

〔그림 3〕 입력장치의 접속회로구성

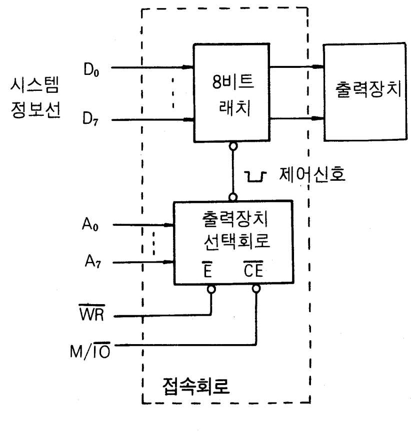



〔그림 4〕 스위치 입력장치의 접속 예




(a) 하드웨어 접속회로 (b) 입력장치의 할당표

〔그림 5〕 출력장치의 접속회로구성



프로세서 시스템으로부터 출력장치에 정보를 내보내기 위해 "OUT port, AL"이라는 명령어를 사용하는데 이때의 제어신호는 M/$\overline{IO}$신호, $\overline{WR}$신호 및 포트로 지정된 주소값이 발생된다.

또한, 출력장치의 접속회로 구성은 3-상태 버퍼가 없이 직접 래치 레지스터에 정보를 받아 출력하는 단순방식으로서 그림 5에 이의 구성을 보여준다.

## 연산처리를 위한 명령어

마이크로프로세서에서 사용되는 수의 개념은 2진수 또는 10진수 인가를 구별하여야 하고, 또한 이들이 부호를 포함한 수인지, 아닌지를 구별하여 수의 표시방식이 정수형(Fixed Point) 또는 부동소숫점(Floating Point)방식인지 구별하여야 한다.

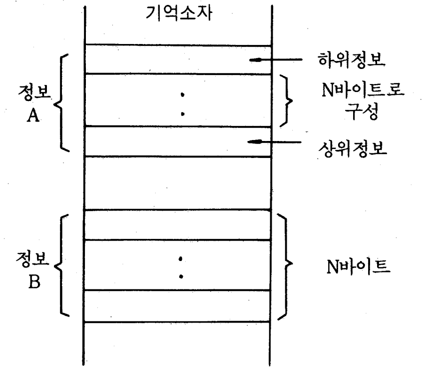
특히, 10진수는 BCD형 10진수인지, 문자형 10진수(ASCII코드)인지에 따라 처리되는 명령어 종류는 다양하다. 그러나 이들 연산의 기본은 2진수 처리로서 가·감·승·제에 대한 명령어는 ADD, SUB, MUL, DIV로 이뤄진다(자세한 내용은 명령어표를 참고하기 바란다).

10진수인 경우에는 2진수 연산방식으로 처리한 후 10진수로 변환해주며, 이는 DAA(Decimal Adjust for Addition) 라는 명령어로 이뤄졌다. 실제 컴퓨터 기억소자에 기억된 소자는 그림 6과 같이 여러 바이트에 나누어 기억된다.

예를 들면 A번지의 정보와 B번지 정보를 더하여 A번지에 결과를 저장하기 위한 명령어(A=A+B)로 낮은 단위의 수부터 더하여 자리수가 높은 상태로 그림 6(b)와 같은 연산처리가 된다.

부동소숫점 방식에서 모든 수는 그림 7과 같이 4바이트로 구성되며 (포트란의 경우), 더욱 정밀한 정보값을 처리하기 위해서는 배정도 방식인 유효수자(Mantissa)를 확장하면 된다.

〔그림 6〕 정수형 연산처리 예

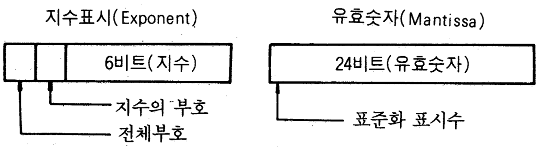



(a) 기억소자의 정보저장내용

```
MBADD : MOV CX, N
        MOV BX, AVGND
        MOV DX, ADDND
        MOV AL, 0
LOOP  : MOV AL, [BX]
        ADC AL, [DX]
        MOV [DX], AL
        INC DX
        INC BX
        DEC CX
        JNZ LOOP
```

(b) N자리 덧셈 프로그램 예

〔그림 7〕 부동소수점 수의 표현방식




〔표 1〕 내부코드와 입출력코드의 변환과정

| | |
|---|---|
| IN AL, 08H | 입력포트(08)로 부터 ASCII코드를 읽어들임. |
| AND AL, 0FH | Masking변환. 즉 내부코드로 변환됨 |
| SHL AL, 4 | 4비트 이동. 즉 상위숫자값 조정 |
| MOV BL, AL | BL에 저장 |
| IN AL, 08H | 2번째 정보를 읽어들임 |
| AND AL, 0FH | 내부코드로 변환시킴 |
| OR AL, BL | 입력된 2개 숫자를 합성하여 2자리 내부숫자를 생성함 |

## 명령어 처리순서의 제어를 위한 명령어

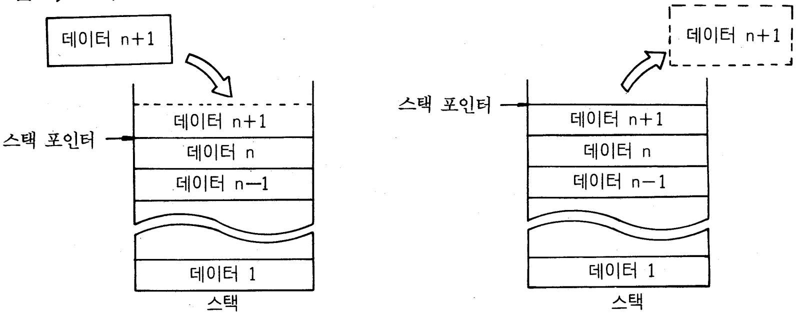
명령어는 기억소자에 저장된 순서에 따라 순차적으로 처리되나 반복 프로그램과 같은 특정상태에 따라 이들 순서를 변경해야 하며, 이를 위한 명령어로는 JUMP와 CALL 등이 사용된다.

이들은 특정조건 즉 플래그(F)비트 상태에 따라 점프의 여부가 결정되는 방식과 무조건 순서를 바꾸는 방법으로 세분되며, 또 현재의 위치부터 256개 이내의 명령어로 이동되는 방식과 직접 주소를 지정하는 방법이 사용된다. 즉 +127개 이내의 이동되는 조건부 명령어는 자리올림(C)상태, 부호(S)상태, 0(Z)상태 등에 따라 결정되며, 2바이트 명령어(OP 코드부와 8비트 비 계산된 점프 주소값)로 비교적 짧은 시간내에 순서가 변경된다.

그러나 직접 주소지정방식은 같은 세그먼트 내에서는 IP값만, 다른 세그먼트의 이동은 CS와 IP값 모두를 변경해 주어야 하기 때문에 명령어 구성이 여러 바이트로 구성된다.

또한, 반복을 위한 명령어는 프로그램상에서 매우 많이 사용되므로 LOOP라는 특별한 명령어를 사용한다. 반복

〔그림 8〕 스택




PUSH 동작 : 스택에 데이터를 격납한다. POP 동작 : 스택에서 데이터를 꺼낸다.

회수는 CX레지스터로 저장하는데 제로 플래그 상태에 따라 반복을 중단할 수 있다. 서브루틴이거나 인터럽트 처리과정에서도 프로그램 순서를 변경해 주어야 하나, 이들은 수행한 후에도 반드시 다시 불러 사용하였던 프로그램 속으로 되돌아가야 하므로 되돌아 가야 할(Return)주소를 기억하고 있어야 한다. 이를 위한 주소방식을 스택(Stack) 메모리라 하며 이의 활용은 매우 중요하므로 잘 살펴볼 필요가 있다.

일반적인 주기억장치가 임의의 주소번지 지정호출(Random Access)방식임에 비하여 스택 기억장치는 마지막에 기억된 정보가 반드시 먼저 호출되는 LIFO(Last In First Out)방식으로 운영된다.

따라서 정보가 들어있는 최상위 위치를 알고 있어야 하는데 여기에는 SP(Stack Pointer)라는 프로세서 내부레지스터를 사용한다. 이 레지스터가 지시하는 번지로부터 정보를 읽어내거나(POP 동작이라 부름), 정보를 저장(PUSH동작)하거나 한다. 이의 하드웨어적인 구성은 일반적으로 RAM 기억소자와 똑같으며 단지 PUSH와 POP 명령어 동작에 의하여 순차적으로 읽고 쓰게 된다(그림 8 참조).

그러면 이들 스택 기억장치를 사용한 부 프로그램(Subroutine)이나 인터럽트를 설명하겠다. 이들은 모두 특정한 단위 프로그램을 저장해 놓고 프로그램상에서 반복적으로 불러다 쓰는 방식으로 운영되기 때문에 프로그램 작성과 수정이 쉽고, 기억소자를 효과적으로 사용 가능하다는 장점이 있어 많이 사용되는 것이다.

그러나, 이때 CALL과 인터럽트의 차이점으로 CALL 명령어는 프로그램상에서 작성자가 처리시점을 알고 있기 때문에 그 프로그램 위치에서 사용가능하지만, 인터럽트는 그 발생시점을 프로그램상에서 예측하지 못하고(다음 하드웨어 설계편에서 세부적으로 설명함) 외부 하드웨어에 의하여 특정 프로그램 루틴의 발생시점을 지정시키는 방식이라는 점을 들 수 있다.

즉, 특정 프로그램을 어떻게 지정하느냐에 따라 차이가 있고 사용 측면에서는 동일하다. CALL이나 인터럽트는 프로그램에서 자주 처리되는 위치에 사용되기 때문에, 처리될 때마다 사용되는 각 변수값을 적절히 전달해 주어야 하며, 이를 패싱파라미터(Passing Parameter)라 부른다.

또한, 처리될 변수값이 적을 때는 내부레지스터를 사용하나, 정보값이 많을 때는 기억장치를 통하여 지정한다. 즉 포인터는 정보위치를 지정하고, 카운터 정보의 갯수를 지정하므로 어느 위치부터 몇개의 정보가 있는지를 부프로그램을 불러다 쓸 때 알려주면 된다.

그러나 이보다 좀 더 빠르게 효율적으로 운영하고자 할 때는 스택 메모리방식을 사용한다.

## 비트처리 및 문자처리를 위한 명령어

논리연산과 같은 명령은 해당되는 비트만의 연산이므로 이를 비트처리라 부르며 부울함수를 처리하기 위한 AND, OR, NOT, XOR, TEST 등과 단위 레지스터 내용을 자체이동시키기 위한 Shift(지정 비트수만큼 시프트) Rotate(데이터를 지정 비트수만큼 회전) 등이 포함된다. 특히 이들 비트처리 명령어는 입·출력 정보에 대한 문자패턴의 변형에 사용된다.

예를 들어 표 1과 같이 입력장치로부터 받아 들여진 ASCII 코드는 내부숫자 코드로 변환하기 위하여 0FH와 AND 시켜 구하며, 컴퓨터 내부에서 처리되는 2진수 코드를 출력에 내보내기 위해서 ASCII 코드로 변환해야 하는데 이는 30H와 OR 동작으로 달성될 수 있다.

문자처리를 위한 명령어는 내부에 기억된 문자패턴 그 자체는 변형시키지 않고 단지 편집과 조합을 목적으로 다른 기억 주소번지로 이동하는 명령어로서 편집기(Text Processing)에 적합한 명령어이다.

여기에는 문자의 이동(MOVS), 문자의 비교(CMPS), 문자의 찾음(SCAS) 및 문자의 지정과 호출(LODS, STOS) 등이 포함되어 있다. 이는 Word Star와 같은 편집기에 활용되고 있으며, 이의 응용은 주변에서 많이 사용되는 명령어이므로 각자 습득하기 바란다.

## 응용설계

지금까지 설명된 명령문을 사용하여 실제 주변기기에 연결된 하드웨어가 어떻게 적용되는지를 몇가지 예를 들어 설명하겠다.

### ● 로보트 언어구현

처음의 예는 로보트 구동을 위한 교

시판(Teaching Box)에 대한 예제로서 0에서 7까지 키로 로보트 각 부분을 작동시키기 위한 동작 키이다. 따라서 특정 키를 치면 해당되는 모듈화된 프로그램을 수행시키게 된다.

교시판의 각 키를 수행시키는 주소번지를 1000번지부터 2바이트씩 할당하여, 여기에 각 키기능을 정의하는 프로그램의 시작 주소번지를 기억시켜 둔다. 그리고 교시키가 입력되면 **그림 9**(b)와 같이 즉시 수행 프로그램으로 연결된다.

한 예로 "3"번 키는 로보트의 X축을 다음에 오는 숫자 키 만큼(단위 cm) 이동시킨다면 로보트 명령어는 "312"라 작성할 수 있고, 이는 X축을 12cm 이동시키라는 것이 된다.

앞의 가정에 따라 3번 키를 수행하기 위한 프로그램의 시작번지는 1006번지의 키 테이블 표에 지정되며, 1006번지와 1007번지에는 수행 프로그램의 실제적인 주소값(본 예에서는 1234번지)인 1234가 기억되어 있고, 즉시 1234번지 수행 프로그램으로 연결된다.

따라서 1234번지에서는 다음에 입력되는 숫자의 값을 읽어 X축 모터에 전송하는 프로그램이 작성되어야만 한다.

## ● 프로그램 제어형 구형 펄스 발생기

앞의 예는 입력장치 접속을 위한 예제였고, 2번째 예로서는 출력장치에 연관된 보기를 들겠다. 즉 출력파형의 시간제어를 위한 활용방안으로 주변 하드웨어 응용에서는 이들 문제가 필수적으로 적용된다고 본다.

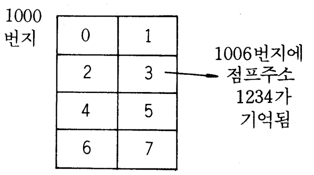
하드웨어 구성은 **그림 10**(a)와 같이 설계하므로 D형 플립플롭을 프로세서의 출력 포트방식에 의하여 "Set", "Clear" 시켜 필요한 시간간격의 펄스폭을 제어할 수 있도록 하였다.

입·출력 명령어에서 이미 설명한 바와 같이 OUT 명령어에 의하여 8086 프로세서는 포트에 해당하는 주소번지와 제어신호($\overline{WR}$, IO/$\overline{M}$)를 발생시키므로 이를 사용하여 디코더에 연결한다.

또한, 발생될 펄스폭의 시간계산은 프로그램을 어떻게 작성하느냐에 따라 결정되며 그림 10(b)와 같이 작성하는 경우의 계산 예는 그림 10(c)에 보여준다.

모든 명령어는 그 명령어를 수행할 때 소요되는 스테이트 수(시스템클럭 수)가 결정되어 있으므로 이를 프로그

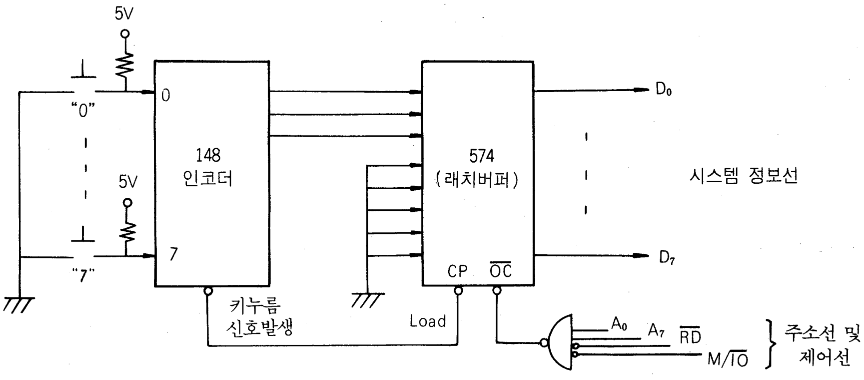
〔그림 9〕 로보트 언어의 구현 예




(a) 교시판

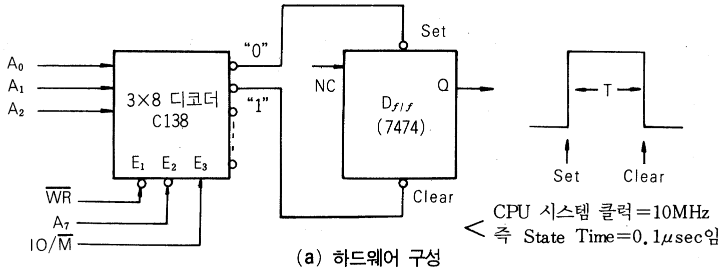
```
BEGIN: MOV BX, TBL  ; BX←1000
       IN  AL, 81H  ; AL←(port81), AL←3
       RCL AX, 1    ; AX←2XAX, AX←0006
       ADD BX, AX   ; BX←BX+AX, BX←1006
       MOV AX, [BX] ; AX←[M]BX, AX←1234
       JMP AX       ; IP←1234
```

(b) 수행 프로그램




(c) 하드웨어 설계

〔그림 10〕 프로그램 제어형 구형펄스 발생기




(a) 하드웨어 구성

| | | | 소요 스테이트 수 |
|---|---|---|---|
| | MOV | BX, N | 4 |
| | OUT | 0H, AL | 10 |
| LOOP: | DEC | BX | 2 |
| | JZ | FIN | 16/4 |
| | JMP | LOOP | 15 |
| FIN: | OUT | 01H, AL | 10 |

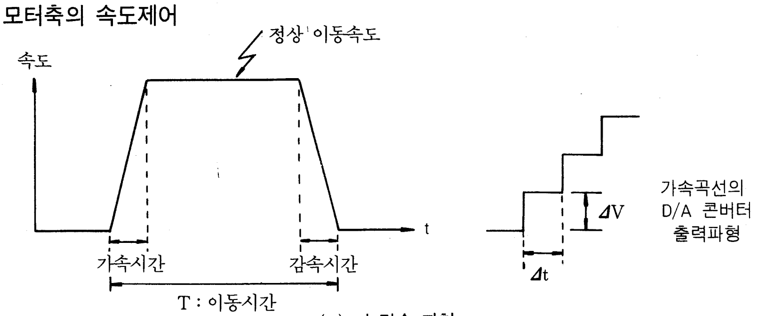
(b) 프로그램과 소요 스테이트 수

$T=[(N-1)\times 21+28]\times 100[nsec]$

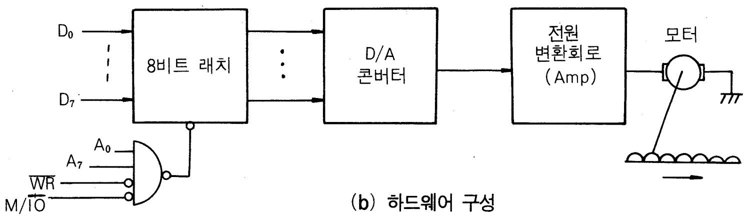
따라서 T=1msec 펄스파형을 위한 근사값 N=476이 됨

(c) 계산 예

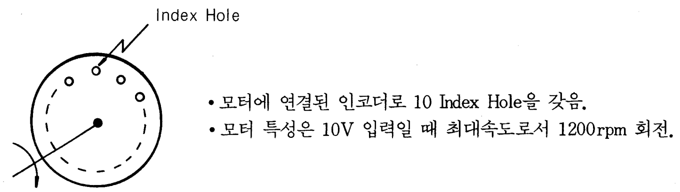
〔그림 11〕 모터축의 속도제어




(a) 가·감속 파형




(b) 하드웨어 구성




(c) 인코더 구성

램 처리과정에 몇개가 사용되는가를 계산하여 시간으로 변환한다.

이때 프로그램상의 "JZ FIN" 명령어에서 스테이트 수 16/4은 점프가 이루어질 때 16개의 시스템클럭이 소요되며, 점프가 이루어지지 않고 다음을 수행할 때는 4개 클럭이 소요됨을 뜻한다. 그림 10(c)의 계산 예에서 시스템 클럭을 10MHz로 사용하면, 즉 스테이트 타임(State Time)이 0.1$\mu$sec가 되므로 1msec 펄스를 만들기 위한 근사값은 N=476이 되며 입력에서 원하는 시간을 입력하면 된다.

## • 모터축의 속도제어

끝으로 실제 사용되고 있는 로보트 축의 속도제어 문제를 예로 들어 설명하겠다.

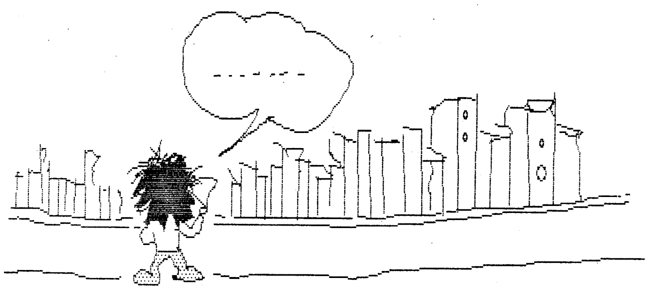
기계장치에 연결된 모터는 정상속도를 얻기 위한 전압을 순간적으로(구형파 모형) 인가하면 연결 축이 손상되므로 그림 11(a)와 같이 서서히 속도를 높여주다가 정지할 때도 서서히 감축시켜야 한다.

이들 가·감속 파형은 기계적 상태에 따라 결정된다. 따라서 프로그램에서 D/A 변환기를 통하여 전압값을 조정해 주어야 한다. 이때 단위 시간당 전압의 크기는 A레지스터(AX 또는 AL) 값을 어떻게 증가시키면서 출력시키느냐에 따라 다르며 그림 11(b)와 같이 하드웨어적으로 구성한다.

실제 로보트에서는 이동시간 및 속도를 내보내며, 위치를 동시에 검출하는 궤환루프(Closed Loop)방식이므로 이때는 모터축에 그림 11(c)와 같은 인코더를 부착시켜 펄스 갯수(이는 이동거리로 계산됨)로 확인하게 된다. 본 예에서는 한 회전당 10개의 인덱스 홀을 갖고 최대 모터속도가 10V에서 1200rpm 이라 할 때 로보트의 최대 이동속도는 초당 200㎜가 된다.

그림 11(a)와 같이 작동하는 실제 로보트 구성 프로그램은 각자 구성해 봄으로써 그 이유를 명확히 이해할 수 있다.

이와 같이 본 호에서는 프로그램이 하드웨어 주변회로에 어떻게 활용되어지는가를 예를 들어 설명하였으며, 다음 호에는 전체적인 시스템동작을 위한 제어 프로그램 즉 모니터 프로그램 구성에 대하여 설명하여 로보트 언어에 대한 독자들의 이해를 돕도록 하겠다.



# 디지탈 집적회로

김정선/한국항공대학 전자과교수

마이크로일렉트로닉스의 집적회로기술(Microelectronics IC Technology)은 회로설계 기준과 방법을 많이 바꾸어 놓았다.

즉 IC화 기술이전에 트랜지스터의 가격은 수동소자들의 값에 비해 고가였다. 그러나 모든 것이 경박단소화로 IC화 기술이 붐을 이루자 트랜지스터의 값은 수동소자인 저항이나 캐패시터의 값에 비해 저렴해졌다. 뿐만 아니라 회로설계에 있어서도 능동소자의 숫자는 점차 최소화되어 갔고 그에 따라 회로가 차지하는 공간 또한 최소화되어 갔다. 그리고 기존에는 각 개인이 직접 회로설계를 하고 제작을 했던 일들을 지금은 IC제작사에서 대행하게 되었다. 따라서 우리는 단지 필요로 하는 회로특성을 제작사로부터 발표된 특성과 대조하여 적합한 부분 IC회로를 선택하기만 하면 된다.

이러한 IC칩은 집적도에 따라 SSI, MSI, LSI, VLSI 등으로 분류된다. 이 분류기준은 대부분 게이트회로와 등가인 회로수에 의하여 분류된다. 예를 들어 MSI는 게이트수가 12~100개정도인 집적도를 갖고 있으며 SSI는 이것보다 작고 LSI나 VLSI는 이보다 게이트수가 큰 집적도를 갖는 IC이다.

또 디지탈 집적회로(Digital IC)는 조합논리 IC와 순서논리 IC로 분류할 수 있다. 먼저 조합논리 IC의 구성요건을 알아보면 다음과 같다.

① 입력논리 전압레벨과 출력논리 전압레벨이 같은 것.

② 논리 전압레벨에 변화없이 스테이지(Stage)의 연속접속이 가능할 것.

③ 각 게이트는 또 다른 수개의 게이트를 구동할 수 있을 것.

④ 주어진 논리레벨을 쉽게 결정할 수 있고 잡음에 여유를 가질 수 있도록 큰 폭의 논리레벨차를 가질 것.

⑤ 회로의 소비전력이 작을 것.

⑥ 다이오드, 저항, 트랜지스터만으로 구성될 것.

이러한 구성요건을 갖춘 전압에 의한 구동형 게이트회로는 여러가지 형태로 분류된다. 즉 저항-트랜지스터 로직(RTL), 다이오드-트랜지스터 로직(DTL), 트랜지스터-트랜지스터 로직(TTL), 에미터결합 로직(ECL), 메탈옥사이드 세미콘덕터(MOS), 콤플리멘탈 MOS(CMOS), 인테그레이티드 인젝션 로직(IIL) 등이 있다.

이와 같은 각각의 게이트들은 IC제작공정 등의 여건에 따라 표준형 게이트를 구성한다. 즉 RTL은 정논리의 NOR게이트, DTL은 정논리의 NAND게이트가 표준게이트이다. 각각의 게이트 표준회로의 예와 동작전압 진리표를 그림 1과 표 1에 나타내었다.

그림 1(a)의 RTL회로에서 입력 $x_1$, $x_2$의 로직레벨 1은 +3.6V가, 로직레벨 0는 접지되므로 0V가 가해진다. 따라서 입력 $x_1$, $x_2$단자가 개방되었을 때는 각 트랜지스터의 베이스전류가 흐르지 않으므로 입력전압은 0이 되어 로직 입력레벨은 0으로, 접지된 상태와 같다. 그러나 TTL회로에서는 이와 반대로 입력단이 개방되었을 때, 로직 입력레벨은 1과 같이 동작한다.

이와 같이 로직심볼이 같은 NAND나 OR 게이트로 이루어졌다 할지라도 TTL이냐 DTL이냐에 따라 입력단의 개방시 입력레벨은 각각 다르다. 즉 게이트의 종류에 따라 각기 입력특성이 다르게 나타난다. 따라서 각각의 회로 형태에 대해 우선 검토를 해야 사용하지 않는 입력단자는 어떻게 처리해야 하는지를 분명히 할 수 있을 것이다.

예를 들어 MOS나 CMOS는 바이폴라 트랜지스터처럼 전류구동형이 아니고 전압구동형이므로 구동전압이 가해지든 접지상태가 되든(즉 로직레벨이 0이든 1이든) 입력전류가 흐르지 않고, 고입력 임피던스이므로 개방시 잡음의 영향을 받기 쉬우니 사용하지 않는 입력단자는 반드시 접지 혹은 전압을 걸어 주어야 한다.

## 로직 IC의 특성 비교

각 형태의 로직 IC들은 여러가지 다른 특성들을 가지고 있으나 사용상 비교할 수 있는 주요 특성은 다음과 같다.

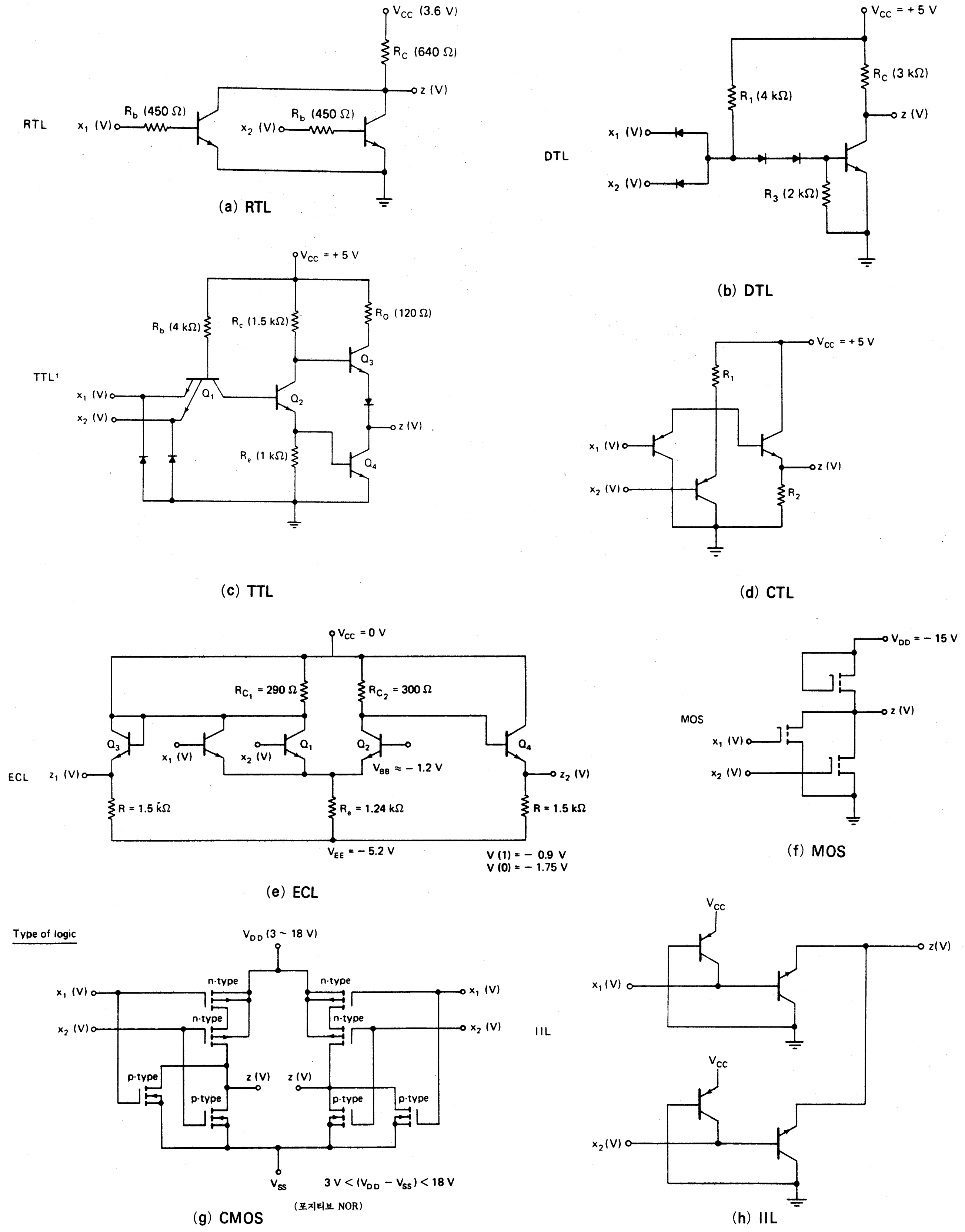
① 스레시홀드 전압(Threshold Voltage)

한 상태에서 또 다른 상태로 출력변화를 주는데 필요한 입력 전압레벨로서 대략 두 상태의 중간 전압레벨이다.

② 동작속도(Operating Speed)

입력레벨의 변화와 출력레벨의 상태변화 사이의 지연시간으로 그 회로의 전달 지연시간이라고도 한다. 일반적

〔그림 1〕 기본 게이트의 회로



으로 한 IC 게이트의 전달 지연시간은 0.5ns~50ns 정도이며 한 로직시스템의 총 지연시간은 게이트당 지연시간에 직렬 게이트의 수를 곱한 값이 된다.

③ 전력소모(Power Dissipation)

디지탈 IC의 전력소비는 회로형태에 따라 게이트당 수 μW~50mW 정도이다. 일반적으로 게이트의 스위칭속도가 높아지면 소비전력도 증가한다. 따라서 소비전력과 스위칭속도 간에는 필요로 하는 요구조건면에서 반대이다.

④ 잡음 여유도(Noise Margin)

대부분의 로직게이트의 입력레벨 변화로 출력상태가 변할때, 0에서 1로 출력상태 변화를 주기 위한 입력 스레시홀드 전압과 1에서 0으로 상태변화를 주기 위한 입력 스레시홀드 전압은 같지 않은데 이 두입력 스레시홀드 전압 레벨의 차이를 일반적으로 잡음 여유도라 한다. 이러한 잡음 여유도는 클수록 잡음에 의한 영향을 적게 받는다.

⑤ 로직 전압 레벨(Logic Voltage Level)

로직레벨의 0과 1을 만들어 주는 전압값은 전력소비, 스위칭 속도, 잡음여유도 등과 같은 시스템 특성에 대해 영향을 준다.

⑥ 팬-인 팬-아웃(fan-in and fan out)

팬-인과 팬-아웃은 동일종류의 게이트를 구동효과에 영향을 미치지 않는 범위에서 연결할 수 있는 게이트의 수로 계산된다. 즉 게이트의 한 입력에 이 입력을 구동하기 위한 동일한 종류의 게이트를 몇개까지 연결 사용할 수 있는가가 팬-인 수이며, 게이트의 한 출력에 동종류의 게이트 입력을 몇개까지 연결하여 구동할 수 있는가를 팬-아웃 수로 계산한다.

⑦ 동작온도(Operating Temperature)

모든 IC는 온도의 변화에 따라 그 특성이 변화되는 반도체로 구성되며 가능한 넓은 온도변화 범위에서도 동작특성의 변화가 적어야 한다.

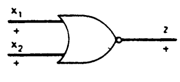
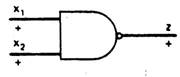
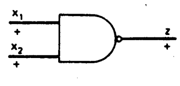
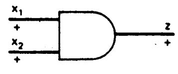
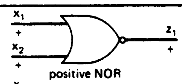
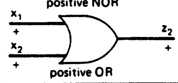
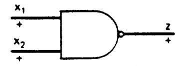
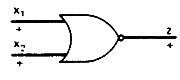
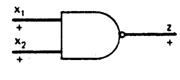
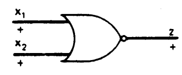
특히 군용으로 쓰이는 IC는 극한상황에서도 기능을 발휘해야 하기 때문에 −55~+125℃에서 사용이 가능해야 하며, 일반용은 0~75℃ 정도이면 충분하다.

중요한 몇가지 게이트 종류와 특성 비교표를 표 2에 나타내었다.

이들의 장단점을 표 3에 보였다.

종류가 다른 IC들을 혼합하여 회로설계를 할 경우 상호간의 호환성에 문제가 될 수 있다. 예를 들어 TTL의 출력은 CMOS를 구동시키는데 적합하지만 CMOS의 출력은 TTL의 입력으로 임피던스의 차이, 구동전류와 출력

〔표 1〕 기본 게이트의 동작 진리치표

| 로직 타입 | 진리표: 전압 | 진리표: 로직 | 심볼 |
|---|---|---|---|
| RTL | $x_1$ (V) / $x_2$ (V) / z (V)<br>0 V / 0 V / −3.6 V<br>0 V / +3.6 V / 0 V<br>+3.6 V / 0 V / 0 V<br>+3.6 V / +3.6 V / 0 V | V (1) = 3.6 V<br>V (0) = 0 V<br>$x_1$ $x_2$ \| z<br>0 0 \| 1<br>0 1 \| 0<br>1 0 \| 0<br>1 1 \| 0 |  positive NOR |
| DTL | $x_1$ (V) / $x_2$ (V) / z (V)<br>+0.5 V / +0.5 V / + 5 V<br>+0.5 V / + 5 V / + 5 V<br>+ 5 V / +0.5 V / + 5 V<br>+ 5 V / + 5 V / +0.5 V | V (1) = 5 V<br>V (0) = 0.5 V<br>$x_1$ $x_2$ \| z<br>0 0 \| 1<br>0 1 \| 1<br>1 0 \| 1<br>1 1 \| 0 |  positive NAND |
| TTL | $x_1$ (V) / $x_2$ (V) / z (V)<br>0 V / 0 V / +5 V<br>0 V / +5 V / +5 V<br>+5 V / 0 V / +5 V<br>+5 V / +5 V / 0 V | V (1) = 5 V<br>V (0) = 0 V<br>$x_1$ $x_2$ \| z<br>0 0 \| 1<br>0 1 \| 1<br>1 0 \| 1<br>1 1 \| 0 |  positive NAND |
| CTL | $x_1$ (V) / $x_2$ (V) / z (V)<br>0 V / 0 V / 0 V<br>0 V / +5 V / 0 V<br>+5 V / 0 V / 0 V<br>+5 V / +5 V / +5 V | V (1) = 5 V<br>V (0) = 0 V<br>$x_1$ $x_2$ \| z<br>0 0 \| 0<br>0 1 \| 0<br>1 0 \| 0<br>1 1 \| 1 |  positive AND |
| ECL | $x_1$ (V) / $x_2$ (V) / $z_1$ (V) / $z_2$ (V)<br>−1.75 V / −1.75 V / −0.9 V / −1.75 V<br>−1.75 V / −0.9 V / −1.75 V / −0.9 V<br>−0.9 V / −1.75 V / −1.75 V / −0.9 V<br>−0.9 V / −0.9 V / −1.75 V / −0.9 V | V (1) = −0.9 V<br>V (0) = −1.75 V<br>$x_1$ $x_2$ \| $z_1$ $z_2$<br>0 0 \| 1 0<br>0 1 \| 0 1<br>1 0 \| 0 1<br>1 1 \| 0 1 |  positive NOR<br> positive OR |
| MOS | $x_1$ (V) / $x_2$ (V) / z (V)<br>0 V / 0 V / −15 V<br>0 V / −15 V / 0 V<br>−15 V / 0 V / 0 V<br>−15 V / −15 V / 0 V | V (1) = 0 V<br>V (0) = −15 V<br>$x_1$ $x_2$ \| z<br>1 1 \| 0<br>1 0 \| 1<br>0 1 \| 1<br>0 0 \| 1 |  positive NAND |
| CMOS | Basic circuit 1:<br>$x_1$ (V) / $x_2$ (V) / z (V)<br>0 V / 0 V / (0.45 ~ 0.55) x $V_{DD}$<br>0 V / (0.45 ~ 0.55) x $V_{DD}$ / 0 V<br>(0.45 ~ 0.55) x $V_{DD}$ / 0 V / 0 V<br>(0.45 ~ 0.55) X $V_{DD}$ / (0.45 ~ 0.55) X $V_{DD}$ / 0 V<br>3 V < $V_{DD}$ < 18 V<br><br>Basic circuit 2.<br>$x_1$ (V) / $x_2$ (V) / z (V)<br>0 V / 0 V / (0.45 ~ 0.55) X $V_{DD}$<br>0 V / (0.45 ~ 0.55) x $V_{DD}$ / 0 V<br>(0.45 ~ 0.55) x $V_{DD}$ / 0 V / 0 V<br>(0.45 ~ 0.55) x $V_{DD}$ / (0.45 ~ 0.55) x $V_{DD}$ / 0 V<br>3 V < $V_{DD}$ < 18 V | V (1) = (0.45 ~ 0.55) x $V_{DD}$<br>V (0) = 0 V<br>$x_1$ $x_2$ \| z<br>0 0 \| 1<br>0 1 \| 0<br>1 0 \| 0<br>1 1 \| 0<br><br>V (1) = (0.45 ~ 0.55) x $V_{DD}$<br>V (0) = 0 V<br>$x_1$ $x_2$ \| z<br>0 0 \| 1<br>0 1 \| 1<br>1 0 \| 1<br>1 1 \| 0 |  positive NOR<br><br> positive NAND |
| IIL | $x_1$ (V) / $x_2$ (V) / z (V)<br>0 V / 0 V / +0.85 V<br>0 V / +0.85 V / 0 V<br>+0.85 V / 0 V / 0 V<br>+0.85 V / +0.85 V / 0 V | V (1) = 0.85 V<br>V (0) = 0 V<br>$x_1$ $x_2$ \| z<br>0 0 \| 1<br>0 1 \| 0<br>1 0 \| 0<br>1 1 \| 0 |  positive NOR |

〔표 2〕 각종 IC의 특성비교

| 로직군 | 기본 로직타입 | 전파시간 (*ns*) | 소비전력 (*mW*) | 노이즈 마진 (*V*) | 팬인 | 최대 팬아웃 |
|---|---|---|---|---|---|---|
| RTL | Positive NOR | 50 | 10 | 0.2 | 3 | 4 |
| DTL | Positive NAND | 25 | 15 | 0.7 | 8 | 8 |
| TTL | Positive NAND | 10 | 20 | 0.4 | 8 | 12 |
| CTL | Positive AND | 5 | 50 | 0.4 | 5 | 25 |
| ECL | Positive OR/NOR | 2 | 50 | 0.4 | 5 | 25 |
| MOS | Positive NAND | 250 | <1 | 2.5 | 10 | 5 |
| CMOS | Positive NOR and Positive NAND | 30 | 0.05 $\mu$W | Depends on $V_{DD}$. (Typically 45% of $V_{DD}$) | 10 | 100 |
| IIL | Positive NOR | 40 | <1 | 0.35 | 16 | 8 |

〔표 3〕 각종 IC의 장단점

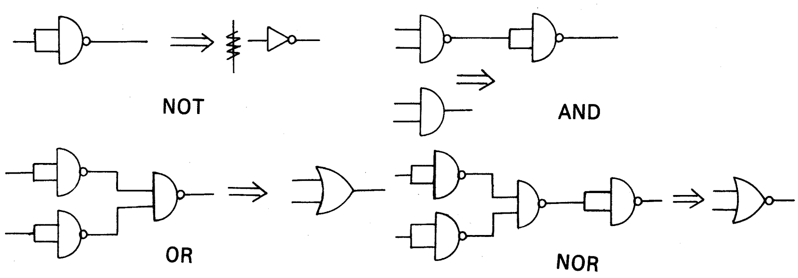
| 논리군 | 장 점 | 단 점 |
|---|---|---|
| RTL | • 저소비전력 | • 불충분한 노이즈마진<br>• 저속 |
| DTL | • 저소비전력 | • 부족한 노이즈마진<br>• 저속 |
| TTL | • 저소비전력<br>• 고속<br>• 고팬아웃 능력<br>• 저가격 | • 타이트한 $V_{CC}$ 허용오차<br>• 순간전력에 민감 |
| CTL | • 더 고속 | • 고가 |
| ECL | • 매우 고속<br>• 고팬아웃 능력<br>• 저노이즈 발생<br>• 보상 출력 | • 다른 로직군과의 인터페이스 어려움<br>• 고가<br>• 회로 레이아웃에 주의 |
| MOS | • 패키지당 회로의 복잡성 증가<br>• 회로기능당 저가<br>• 테스트할 서브시스템 적음<br>• 기능당 파워 드레인 낮음<br>• 스탠다드나 커스텀 생산시 선택 | • 매우 낮은 속도<br>• 다른 군과의 인터페이스와 적절한 동작을 위해 다양한 전력소스가 요구됨 |
| CMOS | • 저속으로 동작할 때 전력소비 낮음<br>• $V_{DD}$의 변화에 무감각<br>• 고노이즈에 면역 | • 저속 |
| IIL | • 매우 높은 패키지 밀도 | |

전류의 차이 등으로 인하여 적합치 않으므로 이 사이에는 특수한 버퍼를 사용해야 한다.

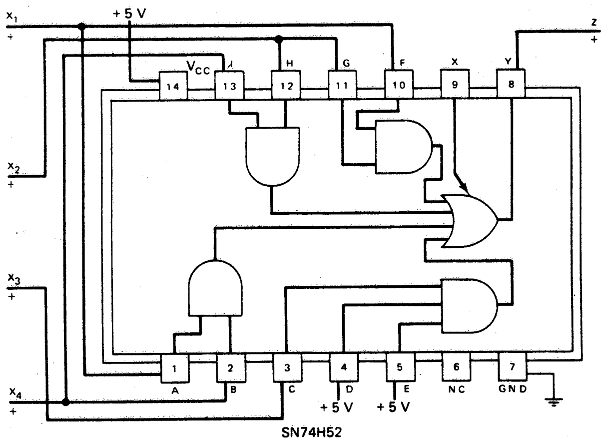
IC중 특히 TTL의 경우는 제작회사에 구분없이 논리기능이 같을 경우에는 같은 명칭으로 표시되며 동일한 기능을 가진 IC가 대부분 IC제작사로 부터 제작 시판되고 있다.

역시 동일기능의 또 다른 종류 즉

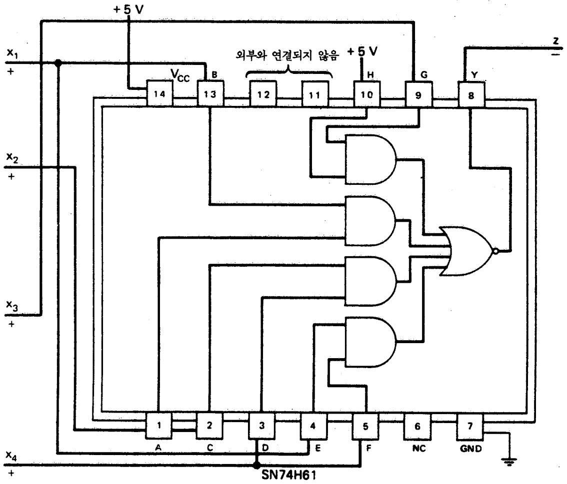
〔그림 2〕 NAND 게이트 4개로 또 다른 로직을 위한 연결방법



〔그림 3〕 IC게이트와 응용예




(a) AND-OR 게이트

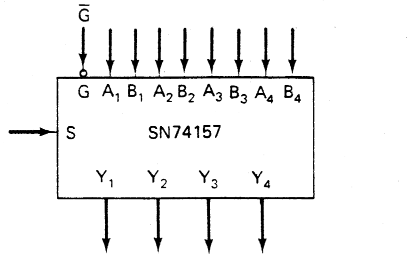



(b) AND-OR-INVERT 게이트

〔표 4〕 TTL/MSI IC

| *MSI* | 기 능 | 디바이스 |
|---|---|---|
| Data selector/multiplexer | 16-bit | 54/74150 |
| | 8-bit, with strobe | 54/74151 |
| | 8-bit | 54/74152 |
| | Dual 4-line-to-1-line | 54/74153 |
| | Quad 2-input | 54/74157 |
| Encoder | Full BCD priority encoder | 54/74147 |
| | Cascadable octal priority encoder | 54/74148 |
| Code converter | 6-line-BCD to 6-line | |
| | Binary, or 4-line to 4-line | 54/74184 |
| | BCD 9's/BCD 10's converters | |
| | 6-bit-binary to 6-bit-BCD converter | 54/74185A |
| Decoder/demultiplexer | BCD-to-decimal (4-to-10 lines) | 54/7442 |
| | Excess-3-to-decimal (4-to-10 lines) | 54/7443 |
| | Excess-3-Gray-to-decimal (4-to-10 lines) | 54/7444 |
| | BCD-to-decimal decoder/driver (4-to-10 lines) | 54/7445 |
| | BCD-to-7 segment decoder/driver | 54/7446 |
| | BCD-to-7 segment decoder/driver | 54/7447 |
| | BCD-to-7 segment decoder/driver | 54/7448 |
| | BCD-to-7 segment decoder/driver | 54/7449 |
| | BCD-to-decimal decoder/driver | 74141 |
| | BCD-to-decimal decoder/driver | 54/74145 |
| | 4-line-to-16-line decoder/demultiplexer | 54/74154 |
| | Dual 2-line-to-4 line decoder/demultiplexer | 54/74155 |
| | Dual 2-line-to-4 line decoder/demultiplexer, open collector | 54/74156 |
| Digital display circuit | BCD-to-decimal decoder/driver | 54/7445, 54/74145, 54/74141 |
| | BCD-to-7-segment decoder/driver | 54/7446A, 54/7447A, 54/74L46 54/74L47, 54/7448, 54/7449 |
| Arithmetic device | Gated full-adder | 54/7480 |
| | 2-bit full-adder | 54/7482 |
| | 4-bit full-adder | 54/7483 |
| | Quad 2-input EXCLUSIVE-OR gate | 54/7486 |
| | 4-bit arithmetic logic unit | 54/74181 |
| | Look-ahead carry generator | 54/74182 |
| Comparator | 4-bit magnitude comparator | 54/7485, 54/74L85, 54/74S85 |

〔그림 4〕 MSI IC의 입출력과 의미

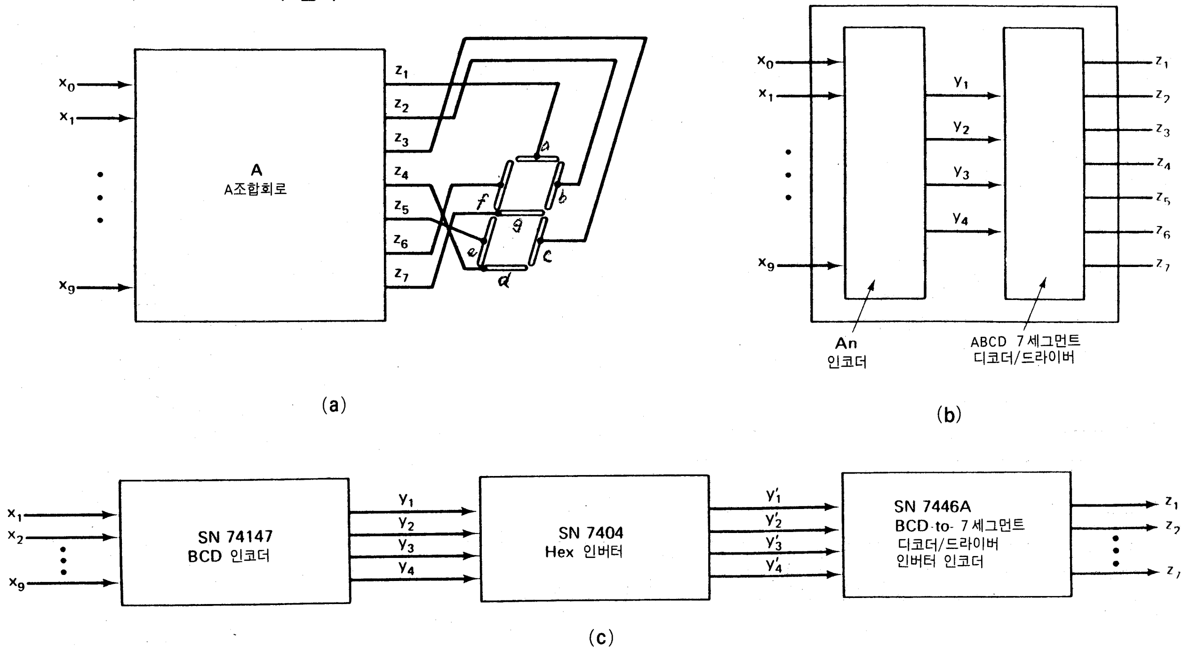
| 레벨 | 의미 | 예 |
|---|---|---|
| A,B,C,... | 입력 |  |
| Y,W | 출력 | |
| $\overline{G}$,$\overline{E}$ | 스트로브(인에이블) 모든 TTL/MSI상에서 Low 액티브 | |
| S | 셀렉트 | |



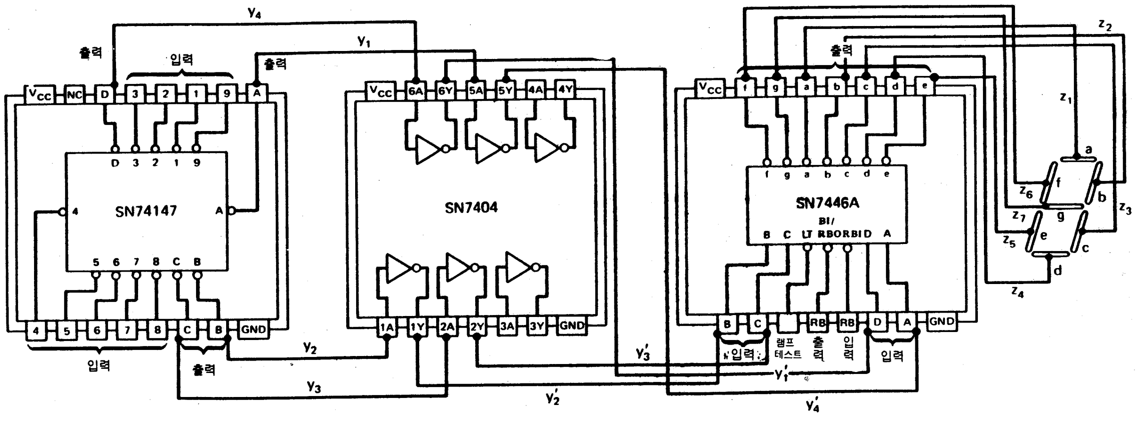
MOS나 CMOS, DTL 등의 IC도 대부분은 같은 명칭들로 표시된다. 예를 들어 TTL의 2입력 4개의 NAND 게이트는 어느 제작사에서나 7400으로 표시되어 나온다.

TTL 게이트류는 74/54 계열이라고 하며, 같은 기능의 TTL에서도 74시리즈와 54시리즈는 온도 특성에 차이가 나는데 74시리즈는 0℃~70℃ 54시리즈는 −55℃~+125℃ 범위이다.

〔그림 5〕 10진수의 7세그먼트 표시 블록도




〔그림 6〕 10진수의 7세그먼트 표시 실제회로 구성



# 로직 IC의 실제와 간단한 응용

디지탈 IC는 우리가 디지탈 회로를 설계할 때 IC 외부와의 연결을 적절히 하여 사용하므로서 설계된 회로가 차지하는 공간이나 소비전력, 경제적인 면 등에서 유리하게 할 수 있다. 예를 들어 7400은 2입력의 NAND 게이트 4개가 내장되어 있다.

이 NAND 게이트를 그림 2와 같이 연결하여 7400 IC칩 한개로 NOR, OR, AND, NOT 등 각 논리 게이트를 구성시킬 수 있다.

NOR게이트 4개를 이용해도 역시 AND, OR, NOR, NOT 게이트 등 어느 회로든 실현이 가능하다. 따라서 NAND 한개와 OR 한개의 회로를 필요로 할 경우 NAND IC와 OR IC를 별도로 구입하여 회로를 구성하는 것보다 NAND 4개가 내장되어 있는 7400 IC 한개를 이용하는 것이 여러가지 면에서 유리하다.

몇가지 게이트 IC의 실제와 응용 예를 그림 3에 나타내었다.

MSI는 SSI보다 다음과 같은 유리한 점이 있다.

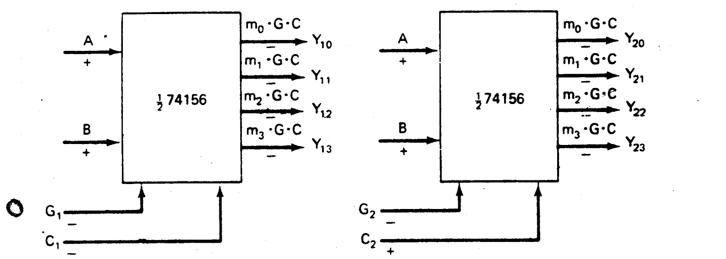
① 집적도를 높임으로서 프린트기판에 SSI 칩을 여러개 배치하여 회로구성을 하는 것보다 구조가 간단해진다.
② 연결(배선)할 부분이 간결해진다.
③ 소비전력 및 발열량이 줄어든다.
④ 가격이 저렴해진다.
⑤ 설계, 고장진단, 수리단가나 시간이 절약 및 감소한다.
⑥ 신뢰도가 높아진다.

몇가지 MSI칩의 블록도를 그림 4에 보였다.

표 4에 MSI 조합논리 IC의 기능과 명칭의 예를 나타내었다.

2진화 10진 코드는 디지탈 회로 혹은 시스템내에서 유용한 데이터로 이용되지만 10진수에 익숙한 우리에게는

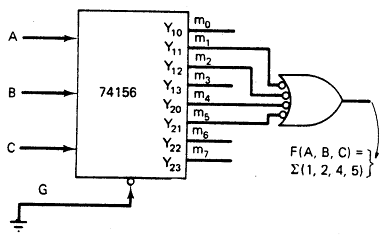
〔그림 7〕 2 라인×4라인 MUX



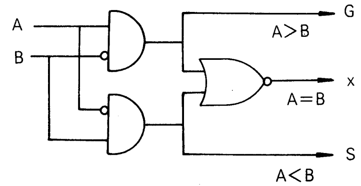

| 입력 | | | | 출력 | | | |
|---|---|---|---|---|---|---|---|
| 스트로브 | 데이터 | 셀렉트 | | | | | |
| $G_1$ | $C_1$ | B | A | $Y_{10}$ | $Y_{11}$ | $Y_{12}$ | $Y_{13}$ |
| H | X | X | X | H | H | H | H |
| L | L | L | L | L | H | H | H |
| L | L | L | H | H | L | H | H |
| L | L | H | L | H | H | L | H |
| L | L | H | H | H | H | H | L |
| X | H | X | X | H | H | H | H |

| 입력 | | | | 출력 | | | |
|---|---|---|---|---|---|---|---|
| 스트로브 | 데이터 | 셀렉트 | | | | | |
| $G_2$ | $C_2$ | A | B | $Y_{20}$ | $Y_{21}$ | $Y_{22}$ | $Y_{23}$ |
| H | X | X | X | H | H | H | H |
| L | H | L | L | L | H | H | H |
| L | H | H | L | H | L | H | H |
| L | H | L | H | H | H | L | H |
| L | H | H | H | H | H | H | L |
| X | L | X | X | H | H | H | H |

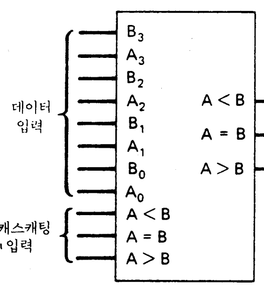
〔그림 8〕 $F(A, B, C) = \Sigma(1, 2, 4, 5)$ 기능 구성도




〔그림 9〕 크기비교회로 (표 6 참조)




〔그림 10〕 두개의 4비트 워드 크기비교 SN7485




(a) 블록도

| 비교입력 | | | | 캐스캐이딩 입력 | | | 출력 | | |
|---|---|---|---|---|---|---|---|---|---|
| A3.B3 | A2.B2 | A1.B1 | A0.B0 | A > B | A < B | A = B | A > B | A < B | A = B |
| A3 > B3 | X | X | X | X | X | X | H | L | L |
| A3 < B3 | X | X | X | X | X | X | L | H | L |
| A3 = B3 | A2 > B2 | X | X | X | X | X | H | L | L |
| A3 = B3 | A2 < B2 | X | X | X | X | X | L | H | L |
| A3 = B2 | A2 = B2 | A1 > B1 | X | X | X | X | H | L | L |
| A3 = B3 | A2 = B2 | A1 < B1 | X | X | X | X | L | H | L |
| A3 = B3 | A2 = B2 | A1 = B1 | A0 > B0 | X | X | X | H | L | L |
| A3 = B3 | A2 = B2 | A1 = B1 | A0 < B0 | X | X | X | L | H | L |
| A3 = B3 | A2 = B2 | A1 = B1 | A0 = B0 | H | L | L | H | L | L |
| A3 = B3 | A2 = B2 | A1 = B1 | A0 = B0 | L | H | L | L | H | L |
| A3 = B3 | A2 = B2 | A1 = B1 | A0 = B0 | L | L | H | L | L | H |

H=고레벨, L=저레벨, X=무관계

(b) 진리표

〔표 5〕 10진수의 7세그먼트 표시 진리표

| $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $z_1$ | $z_2$ | $z_3$ | $z_4$ | $z_5$ | $z_6$ | $z_7$ | Symbol |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 9 |

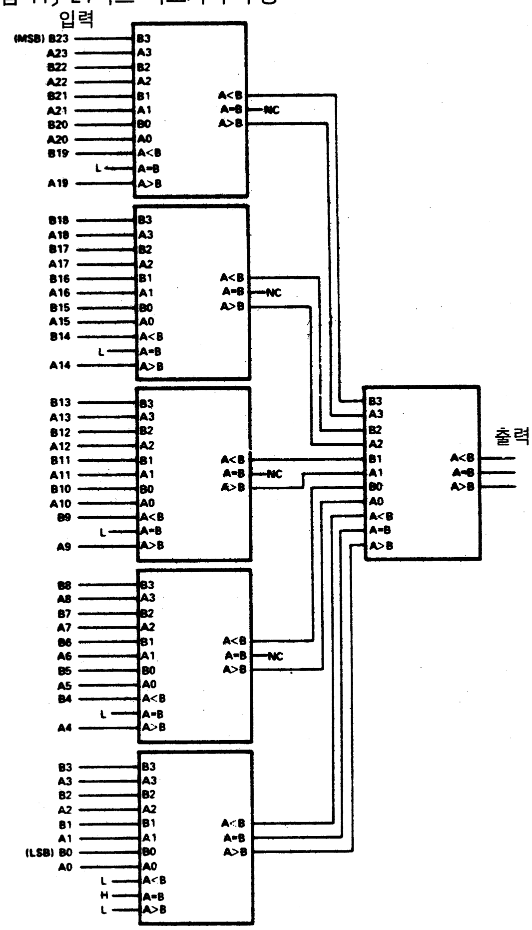
10진과 2진 변환을 실현하는 진리표1 ($x_0$–$x_9$ → $y_1$–$y_4$)

7세그먼트 디코더/드라이버를 실현하는 진리표2 ($y_1$–$y_4$ → $z_1$–$z_7$)

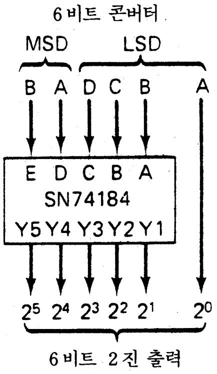
〔표 6〕 크기비교 진리표

| A B | A<B | A=B | A>B |
|---|---|---|---|
| 0 0 | 0 | 1 | 0 |
| 0 1 | 1 | 0 | 0 |
| 1 0 | 0 | 0 | 1 |
| 1 1 | 0 | 1 | 0 |

〔그림 11〕 24비트 비교기의 구성




〔그림 12〕 74184의 블록도와 진리표




| BCD 워드 | 입력 | | | | | | 출력 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | E | D | C | B | A | G | Y5 | Y4 | Y3 | Y2 | Y1 |
| 0-1 | L | L | L | L | L | L | L | L | L | L | L |
| 2-3 | L | L | L | L | H | L | L | L | L | L | H |
| 4-5 | L | L | L | H | L | L | L | L | L | H | L |
| 6-7 | L | L | L | H | H | L | L | L | L | H | H |
| 8-9 | L | L | H | L | L | L | L | L | H | L | L |
| 10-11 | L | H | L | L | L | L | L | L | H | L | H |
| 12-13 | L | H | L | L | H | L | L | L | H | H | L |
| 14-15 | L | H | L | H | L | L | L | L | H | H | H |
| 16-17 | L | H | L | H | H | L | L | H | L | L | L |
| 18-19 | L | H | H | L | L | L | L | H | L | L | H |
| 20-21 | H | L | L | L | L | L | L | H | L | H | L |
| 22-23 | H | L | L | L | H | L | L | H | L | H | H |
| 24-25 | H | L | L | H | L | L | L | H | H | L | L |
| 26-27 | H | L | L | H | H | L | L | H | H | L | H |
| 28-29 | H | L | H | L | L | L | L | H | H | H | L |
| 30-31 | H | H | L | L | L | L | L | H | H | H | H |
| 32-33 | H | H | L | L | H | L | H | L | L | L | L |
| 34-35 | H | H | L | H | L | L | H | L | L | L | H |
| 36-37 | H | H | L | H | H | L | H | L | L | H | L |
| 38-39 | H | H | H | L | L | L | H | L | L | H | H |
| Any | X | X | X | X | X | H | H | H | H | H | H |

H=고레벨 L=저레벨 X=무관계

적합치 않다. 따라서 BCD 코드는 10진 표시방법으로 바꾸어야 우리가 알아보기 쉽게 된다. 한편 10진수의 숫자 모양을 표시하는데는 7세그먼트 표시기를 많이 사용한다. 따라서 BCD코드로부터 이에 상응하는 10진 표시를 하기 위하여는 BCD to 7 세그먼트 변환기가 있어야 하며 이 변환을 위한 동작의 진리치표를 **표 5**에 나타내었으며 0~9의 10개의 10진수에 해당하는 스위치 동작으로부터 7 세그먼트 표시기를 동작시키기 위한 블록도와 실제 IC를 응용한 회로구성도를 **그림 5, 6**에 나타내었다.

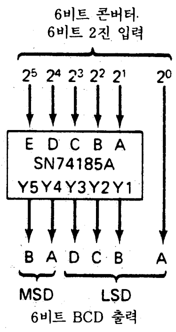
TTL 74156은 2선을 4선으로 보내는 멀티플렉스 회로 2개가 내장되어 있는 IC이며 그 블록도와 동작 진리표는 **그림 7**과 같다.

이 IC를 이용하여 조합논리 회로의 적합한 합회로를 구성할 수도 있다. 예로서 $F(A, B, C)=\Sigma(1, 2, 4, 5)$의 기능회로는 **그림 8**과 같이 연결하면 가능해진다.

이 MUX IC를 2개 이용한 4라인 to 16라인 MUX 회로구성도 가능하다.

두 비트의 크기를 비교하기 위한 진가표는 **표 6**과 같다. 이 진가표로부터

(그림 13) 74185의 블록도와 진리표

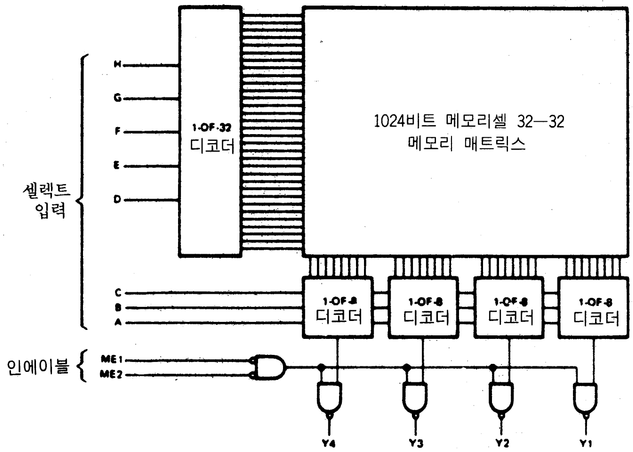



Function Table

| 2진 워드 | 입력 | | | | | | 출력 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2진 셀렉트 | | | | | 인에이블 | | | | | | | | |
| | E | D | C | B | A | G | Y8 | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | Y1 |
| 0-1 | L | L | L | L | L | L | H | H | L | L | L | L | L | L |
| 2-3 | L | L | L | L | H | L | H | H | L | L | L | L | L | H |
| 4-5 | L | L | L | H | L | L | H | H | L | L | L | L | H | L |
| 6-7 | L | L | L | H | H | L | H | H | L | L | L | L | H | H |
| 8-9 | L | L | H | L | L | L | H | H | L | L | L | H | L | L |
| 10-11 | L | L | H | L | H | L | H | H | L | L | H | L | L | L |
| 12-13 | L | L | H | H | L | L | H | H | L | L | H | L | L | H |
| 14-15 | L | L | H | H | H | L | H | H | L | L | H | L | H | L |
| 16-17 | L | H | L | L | L | L | H | H | L | L | H | L | H | H |
| 18-19 | L | H | L | L | H | L | H | H | L | L | H | H | L | L |
| 20-21 | L | H | L | H | L | L | H | H | L | H | L | L | L | L |
| 22-23 | L | H | L | H | H | L | H | H | L | H | L | L | L | H |
| 24-25 | L | H | H | L | L | L | H | H | L | H | L | L | H | L |
| 26-27 | L | H | H | L | H | L | H | H | L | H | L | L | H | H |
| 28-29 | L | H | H | H | L | L | H | H | L | H | L | H | L | L |
| 30-31 | L | H | H | H | H | L | H | H | L | H | H | L | L | L |
| 32-33 | H | L | L | L | L | L | H | H | L | H | H | L | L | H |
| 34-35 | H | L | L | L | H | L | H | H | L | H | H | L | H | L |
| 36-37 | H | L | L | H | L | L | H | H | L | H | H | L | H | H |
| 38-39 | H | L | L | H | H | L | H | H | L | H | H | H | L | L |
| 40-41 | H | L | H | L | L | L | H | H | H | L | L | L | L | L |
| 42-43 | H | L | H | L | H | L | H | H | H | L | L | L | L | H |
| 44-45 | H | L | H | H | L | L | H | H | H | L | L | L | H | L |
| 46-47 | H | L | H | H | H | L | H | H | H | L | L | L | H | H |
| 48-49 | H | H | L | L | L | L | H | H | H | L | L | H | L | L |
| 50-51 | H | H | L | L | H | L | H | H | H | L | H | L | L | L |
| 52-53 | H | H | L | H | L | L | H | H | H | L | H | L | L | H |
| 54-55 | H | H | L | H | H | L | H | H | H | L | H | L | H | L |
| 56-57 | H | H | H | L | L | L | H | H | H | L | H | L | H | H |
| 58-59 | H | H | H | L | H | L | H | H | H | L | H | H | L | L |
| 60-61 | H | H | H | H | L | L | H | H | H | H | L | L | L | L |
| 62-63 | H | H | H | H | H | L | H | H | H | H | L | L | L | H |
| All | X | X | X | X | X | H | H | H | H | H | H | H | H | H |

H = high level, L = low level, X = irrelevant

(그림 14) MSI/TTL SN74187 1024비트 ROM의 2부분




(표 7) TTL/SSI 플립플롭

| 플립플롭 | 디바이스 |
|---|---|
| Edge-triggered J-K | 54/7470 |
| J-K master–slave | 54/7472 |
| Dual J-K master–slave | 54/7473 |
| Dual D-type edge-triggered | 54/7474 |
| Dual J-K master–slave, preset and clear | 54/7476 |
| J-K master–slave | 54/74104 |
| J-K master–slave | 54/74105 |
| Dual J-K master–slave | 54/74107 |
| Monostable nonretriggerable | 54/74121 |
| R-S master–slave | 54/74L71 |
| J-K master–slave | 54/74L72 |
| Dual J-K master–slave | 54/74L73 |
| Dual D-type edge-triggered | 54/74L74 |
| Dual J-K master–slave | 54/74L78 |
| J-K master–slave | 54H/74H71 |
| J-K master–slave | 54H/74H72 |
| Dual J-K master–slave | 54H/74H73 |
| Dual D-type edge-triggered | 54H/74H74 |
| Dual J-K master–slave with preset and clear | 54H/74H76 |
| Dual J-K master–slave | 54H/74H78 |
| J-K, negative edge-triggered | 54H/74H101 |
| J-K, negative edge-triggered | 54H/74H102 |
| Dual J-K, negative edge-triggered | 54H/74H103 |
| Dual J-K, negative edge-triggered | 54H/74H106 |
| Dual J-K, negative edge-triggered | 54H/74H108 |

설계된 논리회로는 그림 9와 같이 된다.

이 원리를 이용하여 여러 비트로 구성되는 2진수 등의 비교기를 설계할 수 있다. 예를 들어 4비트로 구성되는 2개의 2진수 $A(A_3A_2A_1A_0)$, $B(B_3B_2B_1B_0)$의 크기를 비교할 수 있는 비교기를 설계할 경우, 우선 그림 9를 이용하여 A, B가 같은 조건은 $(A=B)=x$인 $(A_3=B_3)=x_3$, $(A_2=B_2)=x_2$, $(A_1=B_1)=x_1$, $(A_0=B_0)=x_0$에서 $x_3=1$, $x_2=1$, $x_1=1$, $x_0=1$이어야 A=B가 된다.

즉

$$x_3x_2x_1x_0=1$$

이어야 한다. 그리고 A〉B가 되려면

$$(A>B)=A_3\overline{B}_3+x_3A_2\overline{B}_2+x_3x_2A_1\overline{B}_1+x_3x_2x_1A_0\overline{B}_0$$

가 되어야 한다. 또한 A〈B이려면

$$(A<B)=\overline{A}_3B_3+x_3\overline{A}_2B_2+x_3x_2\overline{A}_1B_1+x_3x_2x_1\overline{A}_0B_0$$

가 되어야 한다. 이 3개의 논리식을 논리회로로 표현하면 4비트 크기의 숫자 비교기가 된다. 이 원리는 예를 들어 2 숫자 A=76541과 B=76321을 비교할 때 우리는 가장 큰자리(MSB;A의 7과 B의 7)부터 비교하여 두수가 같으면 다음 자리(A의 6과 B의 6)를 비교하고 다시 같으면 또 다음자리(A의 5와 B의 3)를 비교하여 A가 크면 (5〉3) A가 크다고 판정한다. 이러한 순서와 원리를 생각하면 앞에서 논의된 비교기의 원리를 쉽게 이해할 수 있을 것이다. 이와 같은 비교회로의 IC가 7485이다. 이 7485 IC의 블록도와 그 동작 진가표를 그림 10에 보였다.

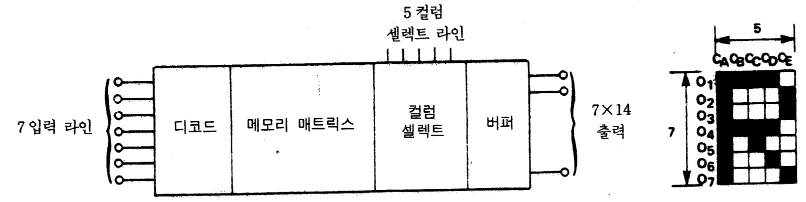
7485 IC는 4비트 크기의 수를 비교하는데 적합하다. 그러나 그 이상의 큰 수를 비교하려면 그림 11과 같이 구성시키므로서 비교할 수 있는 비트 수를 확장할 수가 있다.

BCD와 2진수는 10진수 9이상의 크기에서 부터 서로 달라지며 상호변환이 필요하게 된다. 이러한 용도의 IC가 74184(BCD로부터 2진수로 변환), 74185(2진수로부터 10진코드로 변환)이다. 이들의 구성도와 진리표는 그림 12, 그림 13 에 보였다.

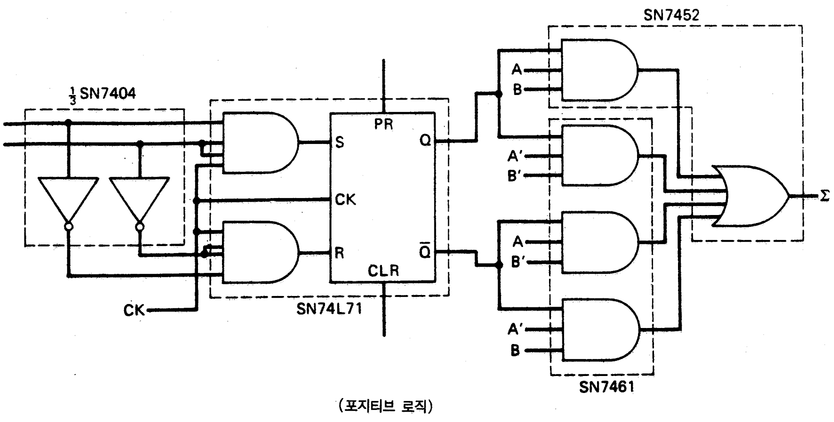
조합논리 회로인 LSI IC 역시 여러 종류가 있는데 대표적으로 ROM IC를 들 수가 있다. ROM IC는 어드레스 디코더 부분과 메모리 부분으로 나누어 지며, 디코더나 메모리 부분의 기본 원리는 전술한 바와 같고 이들의 조합인 ROM의 구성도는 그림 14와 같다.

ROM으로 가장 많이 이용되는 것중

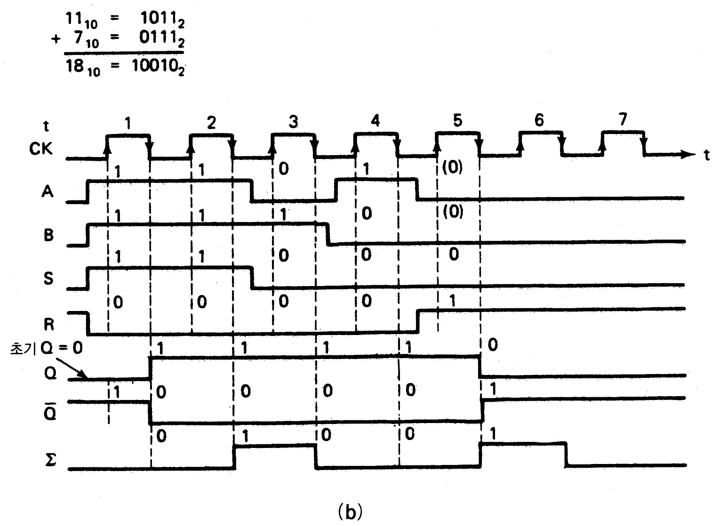
〔그림 15〕 MSI/TTL, LSI/MOS IC 예




〔그림 16〕 S-R FF를 이용한 2진 직렬 가산기

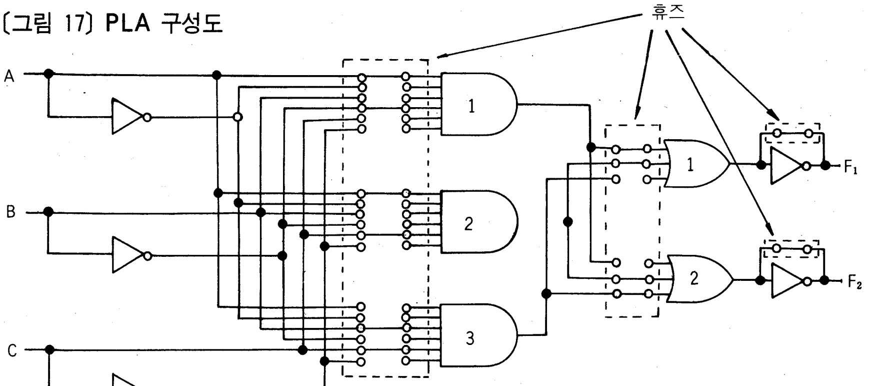



(a)

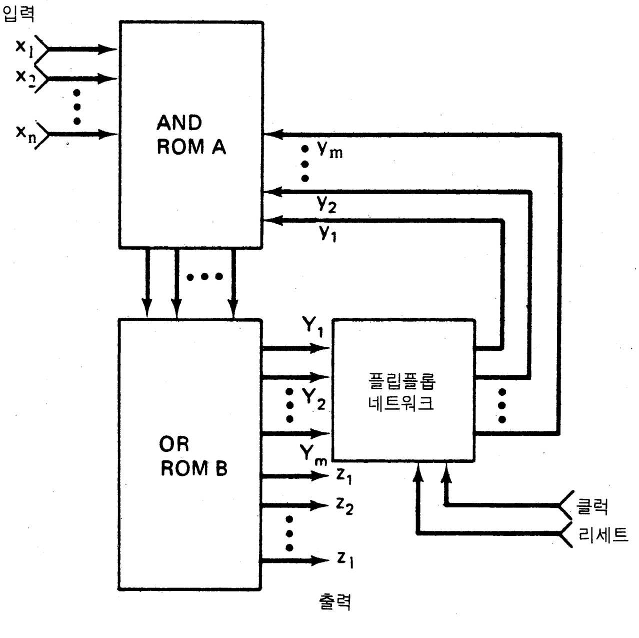



(b)

〔그림 17〕 PLA 구성도




〔그림 18〕 플립플롭을 포함한 PLA




주의 : $y_1, y_2, \cdots, y_m$은 현재상태의 2 진변수
$Y_1, Y_2, \cdots, Y_m$은 다음상태의 2 진변수

의 하나가 문자 발생용 ROM이다.

이와 같은 고집적 회로는 TTL보다 MOS형 IC가 많다. 그림 15에 MSI / TTL과 LSI / MOS IC의 블록도를 보였다.

## 순서논리회로 IC

순서논리회로 IC는 주로 각종 플립플롭 회로의 집합회로로 구성되어 있다. 이들 플립플롭 회로와 조합논리 회로가 혼합되어 계수회로, 비트 시프트 레지스터 회로 등을 구성하는 IC들도 있다.

TTL / SSI 플립플롭 회로 IC들을 표 7에 나타내었다.

순서논리 IC를 이용한 2진 직렬 가산기 회로구성 예를 그림 16에 보였다.

대부분의 디지탈 IC들은 제작회사에서 제품화될 때 이미 그 성능 및 용도가 정해지기 때문에 IC 내부의 내용을 변경시키는 것이 불가능하다. 그러나 PROM이나 PLA(Programmable Logic Array) 같은 것은 사용자에 의하여 단 한번의 프로그램으로도 IC의 내부 회로형태나 기억상태를 변경시킬 수 있다.

주로 AND, OR, NOT의 기본 게이트 조합기능으로 구성되는 가장 기본적인 PLA는 AND-OR-ROM의 매트릭스로 근래에는 궤환로에 플립플롭 회로를 포함하는 것도 있다.

이러한 PLA IC는 IC 명칭에 의하여 성능과 기능이 확정되는 일반 IC와는 달리 그 실제 내용은 사용자에 의하여 변형되며 IC 명칭만으로는 기능이 미지수인 관계로 사용자만이 정확한 기능을 알 수 있으므로 비밀보호 등에 유리하다.

간단한 PLA의 구성도를 그림 17에 보였다. 여기에서 휴즈부분은 사용자에 의하여 프로그램이 가능한 부분으로 휴즈의 절단에 의해 프로그램이 된다. 플립플롭을 포함하는 PLA 블록도 예는 그림 18에 보였다.

이외에도 IC는 그 집적기술이 점차 발전하면서 VLSI화 되어가고 그 기능은 하나의 부품에서 부터 시스템까지 커버가 가능하게 되고 있다. 또한 누구나 공통 기능의 IC를 선택하여 회로를 설계하는 방식에서부터, 설계된 시스템이나 부분 회로를 IC화 하는 주문형 IC(고유목적용 IC )로 변화해 가고 있는 추세이다.

# 텍스트화일을 프린터로 출력해주는 유틸리티
# TPRINT

양왕성/ 성균관대 마이컴연구회

**텍스트 화면을 출력하고자 할때는 한글이 제대로 출력되지 않거나 그래픽 캐릭터가 제대로 출력되지 않는 등의 여러가지 문제점에 부딪히게 된다. 여기에 소개하는 300라인 정도 밖에 안되는 짧은 프로그램을 이용하면 이러한 문제점을 깨끗이 해결하고 원하는 출력을 얻을 수 있다.**

컴퓨터의 주변장치 중에 고가격이라는 특성으로 본체의 보급율에는 미치지 못했던 프린터가 최근 다시 일어나는 컴퓨터 붐으로 그 이용도가 증가하고 있다. 그리하여 컴퓨터를 사용하는 장소가 다양해지고 프린터를 쓰는 곳이 많아짐에 따라 이제 프린터는 쉽게 우리 주변에서 접할 수 있게 됐다.

그러나 가끔 프린터가 마음먹은 대로 작동하지 않을 때는 사용하기가 여간 답답하지 않다. 이러한 이유 때문에 필자는 일반적으로 많이 사용하는 텍스트화일을 프린터로 출력하는 유틸리티 프로그램을 작성해 보았다.

## 텍스트화일의 프린트출력시 문제점

만약 TEST.TXT라는 화일을 프린터로 출력하려면 도스상에서 다음과 같이

```
C:\>TYPE TEST.TXT > PRN
```

하거나

```
C:\COPY TEST.TXT PRN
```

라고 하면 간단히 프린터 출력을 얻을 수 있다. 또 위의 방법보다는 못하지만 Ctrl-P로도 프린터 출력을 얻을 수 있다.

여기에 기능을 추가하여 페이지마다 번호를 준다든지 각 줄마다 라인번호를 준다든지 하고 싶을때는 도스 디스켓의 PRINT.COM이나 노턴유틸리티의 LP.EXE를 이용하면 어느정도 원하는 출력을 얻을 수 있게 된다.

그러나 위에서 설명한 방법으로 프린트를 한다고 해도 탭의 위치가 바르게 찍히지 않거나 129 이상의 그래픽캐릭터가 제대로 출력되지 않는 등의 문제점이 발생한다. 또 제대로 출력된다고 하더라도 크기가 제대로 맞지 않고, 한글이 제대로 출력되지 않는 등의 문제점에 부딪치게 된다.

이러한 문제에 대한 해결책으로는 세운한글카드를 사용하고 있을 경우 HSKEY를 실행시키면 그래픽코드 출력과 한글 출력의 문제점은 어느정도 해결할 수 있다.

그러나 탭코드 문제와 그래픽코드가 두배로 확대되어서 나오는 문제는 해결할 수 없다. 또 이러한 문제 외에도 화면상의 31이하의 특수 캐릭터들을 출력 못한다든지 실제화일은 무척 긴데 $1A, 〈, ^Z, :, EOF, 〉 등의 코드를 만나면 그 뒷부분은 출력이 안된다든지 유틸리티를 사용하여 여러 페이지를 프린트할 경우 몇장을 찍다가 중간 정도에서 망쳐 버렸을 때는 다시 처음으로 찍어야 하는 등 여러가지 문제에 부딪치게 된다.

## TPRINT의 특징

TPRINT는 위에서 언급한 문제들을 해결한 프로그램으로 처음에는 그래픽코드 문제를 해결하기 위해 터보 파스칼 버전4.0에서 작성된 100라인 정도의 조그마한 프로그램이었으나 서너달 동안 쓰면서 발생했던 여러가지 문제를 해결하는 기능들이 추가되면서 300라인 정도의 프로그램이 되었다.

이 프로그램이 추구하는 것은 편집할 때의 화면 그대로 인쇄하기 위한 것으로 그 특징은 다음과 같다.

1. 화면에서 나타난 캐릭터는 프린터에 그대로 출력된다.

예를 들면 화살표 문자나 웃는 얼굴 문자 등 영문편집기(Editer)로 편집할 때 화면에는 나오는데 프린터로는 출력되지 않는 경우를 여러차례 경험했을 것이다. 이 프로그램에서는 프린터의 화면에 있는 캐릭터가 프린터로 그대로 출력된다.

2. $1A, 〈, ^Z, :, EOF, 〉 코드에 관계없이 화일을 모두 프린트한다.

예를 들어 $1A 코드는 화면상에서 오른쪽 화살표인데 그대로 프린트한다.

3. 탭코드($09)에 의해 좌우정렬이 흔들리지 않는다.

세로줄 그래픽캐릭터로 디자인되어 있는 화일을 인쇄할 때도 줄이 똑바로 프린트된다.

4. 여러 페이지를 프린트할 때 필요한 페이지를 선택적으로 프린트할 수 있다.
물론 페이지나 라인번호를 줄 수도 있다.

## 사용방법

TPRINT를 사용하려면 다음과 같이

```
C:\>TPRINT  filename  [ /n /c /s /LX /136 /g ]
```

라고 하면 된다. 여기서 화일이름 뒤에 붙이는 옵션의 기능은 다음과 같다.

- /n: 헤더(filename과 page번호)를 찍지 않는다.
- /c: 라인 번호를 준다.
- /LX: LX 기종에서 그래픽코드를 출력할 때
- /s: 축소 모드로 인쇄한다.
- /136: 136 컬럼 프린터시
- /G: 한글이 없이 영문과 그래픽코드만 필요할 때

그리고 TPRINT 다음에 화일이름없이 실행하면 옵션에 대한 설명이 나온다. 옵션은 복합옵션이 가능하나 /LX 옵션은 /s, /136 옵션과 같이 쓸 수 없다. 그리고 /G 옵션도 /s 옵션과 같이 쓸 수 없다.
실행시키면 화면상에 다음과 같은 메시지를 출력한다.

```
1.Printing-page  2.Skip-page  3.Continue-print  0.Quit
[  1 Page ] Select >
```

이때 1, 2, 3, 0 또는 P, S, C, Q 등의 키중에 하나를 선택하면 되는데 각 키의 기능은 다음과 같다.

- P: 1페이지를 프린트하고 다음 페이지도 같은 형식으로 묻는다.
- S: 1 페이지를 건너 뛰고 다음 페이지도 같은 형식으로 묻는다.
- C: 다른 키가 눌려질 때까지 계속 프린트한다.
- Q: 끝 마친다.

그리고 화일내에서 제어할 수 있는 명령어가 있다. 첫 컬럼이 "." 즉, 도트로 시작되면 그 행은 프린트하지 않는다. 또 ".new"로 시작되면 새로운 페이지로 폼피드한다. 그리고 ESC 코드(27 : $1B) 뒤에 128보다 작은 값의 캐릭터가 오면 ESC 프린터 제어코드로 인식을 하고 그렇지 않으면 그래픽 캐릭터 왼쪽 화살표로 프린트한다.

이 프로그램의 실행 예에 대한 이해를 돕기 위해 화면 1에는 HSKEY를 올리고 프린터를 다음과 같이

```
print chr$(27)+"h"+chr$(27)+"W"+chr$(27)+"V"
```

라고 세팅한 후 프린트로 출력시킨 예를 보인다.

〔화면 1〕 **Print chr$(27)+"h"+chr$(27)+"W"+chr$(27)+"V"로 세팅하여 출력한 예**

```
<*>                                                          1989년 4월 24
일
    Text-Print Document
                                                                || 예 (Exa
mple) ||
  File  :   TPRINT   EXE    12784    4-24-89    3:35p

  사용법:   TPRINT  filename  [ /n /c /s /132 /LX /g ]       ſ ½
            ------  ---------  -------------------------     α² ÷ β² ≡ φ
      * filename--프린트할 TEXT 화일                           ∫ ¼
      * 옵션------/n ... Header not print        기본 Set
                  /c ... line-number Count
                  /132.. 132 column printer  ────80 column   ≈≈≈≈≈≈≈≈≈≈≈ ±
                  /LX... LX-..9 pin printer  ────LQ-24 pin    Σ (ᅟ π ⁿ ) ≤ ∞
                  /s ... Small-mode printing
                  /g ... English Graphic-mode                   ∘ ∘ ∘ ∘

  .     <Dot> 1 Column 에 점 이 오면 print 되지 않는다.     ─┼─
  .new  새 Page 로 넘긴다.

    화면에 편집/보는 그대로 프린트를 할 수 있읍니다.        editer : PE2
    세운-한글-카드에서 한글과 그래픽코드로 편집한 화일을
  Epson 계열의 printer 로 출력하는 화일입니다.      <Yang>   ☎ 274-2452
```

〔화면 2〕 TPRINT TPRINT.DOC/n을 실행하여 출력한 예

```
<*>                                                    •••• 1989년 4월 24일
 Text-Print Document                                    예 (Example)
 File  :  TPRINT  EXE   12784   4-24-89   3:35p
 사용법:  TPRINT  filename  [ /n /c /s /132 /LX /g ]     ⌠½
          ------  --------  ------------------------    │ α² ÷ β² ≡ φ
    * filename--프린트할 TEXT 화일                       ⌡¼
    * 옵션------/n ... Header not print      기본 Set
                /c ... line-number Count                ☻♥♦♣♠☻
                /132.. 132 column printer  ◄─80 column   ≈≈≈≈≈≈≈≈≈≈ ±
                /LX... LX-..9 pin printer  ◄─LQ-24 pin   Σ(µπⁿ) ≤ ∞
                /s ... Small-mode printing
                /g ... English Graphic-mode              ° ° ° °  § § §
                                                            ▲      ↑↑↑
 .      <Dot> 1 Column 에 점 이 오면 print 되지 않는다.    ◄─┼─►  ↔↔↔↔↔↔↔
 .new   새 Page 로 넘긴다.                                  ▼      ↓↓↓

   화면에 편집/보는 그대로 프린트를 할 수 있읍니다.       editer : PE2
   세운-한글-카드에서 한글과 그래픽코드로 편집한 화일을    ▒▒▒▒████████████
  Epson 계열의 printer 로 출력하는 화일입니다.   <Yang>    ☎ 274-2452
```

그리고 화면 2에는 똑같은 내용을

TPRINT TPRINT.DOC / n

으로 실행시켜 나타낸다. 이 두 화면을 비교하게 되면 TPRINT의 특징을 알 수 있을 것이다.

## 알고리즘

이 프로그램의 알고리즘은 다음과 같다. (리스트 1 참조)

첫째, Procedure Lineread(116라인-145라인)에서는 한 행을 화일에서 읽어 내면서 화면상의 특수 캐릭터를 판별하여 프린터 코드로 바꿔주는 작업을 한다.

화일을 읽을 때 1부터 31까지의 캐릭터중 탭코드, 라인피드코드, 캐리지리턴 코드를 제외한 나머지 캐릭터에 128을 더한다. 그 이유는 화면상의 1-31까지의 캐릭터 테이블은 프린터의 129부터 159까지의 캐릭터 테이블과 일치한다. 화면상의 129부터 159까지의 테이블은 다행히도 우리와는 무관한(?) 캐릭터 집합이므로 프린터로 출력되지 않아도 무관하기 때문이다(표 1, 2 참조).

그리고 ESC 코드는 그뒤에 그래픽코드가 나오면 왼쪽 화살표코드로 인식하고 그렇지 않으면 ESC 프린터 제어코드로 인식하여 그대로 통과한다. 이렇게 해서 화면상의 특수캐릭터가 프린터로 출력된다.

둘째, Procedure Encoding(189라인-214라인)은 화면상의 캐릭터 비율 그대로 인쇄하기 위한 루틴이다.

여기서는 연속되는 그래픽코드 앞뒤에 축소코드와 축소해제코드를 넣어서 그래픽코드가 두배로 늘어져서 나오는 것을 막는다. 그러나 LQ 프린터는 한 라인상에서 특정문자의 축소 확대가 자유로우나 LX 프린터는 한 라인상에서 축소코드를 써서 특수문자만 축소할 수 없다.

LX 프린터에도 꼭같은 기능을 부여하려고 연구하던중 LX 프린터는 확대코드를 써서 특정문자만 확대할 수 있다는데 주목을 하고 LX 프린터로도 같은 기능을 갖도록 /LX 옵션을 추가하였다.

/LX 옵션이 들어오면 처음부터 프린터를 축소모드로 바꾼 다음에 그래픽 캐릭터는 축소모드 그대로 프린트하고 일반문자는 축소모드에서 절반 크기로 나오는데 이것을 두배 확대시켜 출력해 균형을 잡았다. 즉 일반문자 앞과 뒤에 확대코드와 확대 해제코드를 넣었다. /LX 옵션을 사용하면 속도는 조금 느려지나 보통모드에서 프린트한 것보다 문자들이 훨씬 예쁘게 프린트 되는데 LQ 프린터에서 사용해도 인쇄상태가 보기좋다.

세째, Procedure Tabcontrol(146 line~163 line)에서는 탭코드에 의해 좌우 정렬이 맞지 않는 현상을 방지한다.

일반적인 편집기는 저장메모리를 줄이기 위해서 스페이스(spacé : 32 : $20)가 연속으로 나열되면 8배수의 컬럼으로 계산하여 탭코드로 바꿔서 저장하는데 이 코드를 프린터에

〔표 1〕 화면상(1～31)
프린터(128～159)

| | | | |
|---|---|---|---|
| DECIMAL VALUE | ➡ | 128 | 144 |
| ⬇ | HEXA DECIMAL VALUE | 8 | 9 |
| 0 | 0 | BLANK (NULL) | ► |
| 1 | 1 | ☺ | ◄ |
| 2 | 2 | ☻ | ↕ |
| 3 | 3 | ♥ | ‼ |
| 4 | 4 | ♦ | ¶ |
| 5 | 5 | ♣ | § |
| 6 | 6 | ♠ | ▬ |
| 7 | 7 | • | ↨ |
| 8 | 8 | ◘ | ↑ |
| 9 | 9 | ○ | ↓ |
| 10 | A | ◙ | → |
| 11 | B | ♂ | ← |
| 12 | C | ♀ | ∟ |
| 13 | D | ♪ | ↔ |
| 14 | E | ♫ | ▲ |
| 15 | F | ☼ | ▼ |

〔표 2〕 화면상(128～159)

| | | | |
|---|---|---|---|
| DECIMAL VALUE | ➡ | 128 | 144 |
| ⬇ | HEXA DECIMAL VALUE | 8 | 9 |
| 0 | 0 | Ç | É |
| 1 | 1 | ü | æ |
| 2 | 2 | é | Æ |
| 3 | 3 | â | ô |
| 4 | 4 | ä | ö |
| 5 | 5 | à | ò |
| 6 | 6 | å | û |
| 7 | 7 | ç | ù |
| 8 | 8 | ê | ÿ |
| 9 | 9 | ë | Ö |
| 10 | A | è | Ü |
| 11 | B | ï | ¢ |
| 12 | C | î | £ |
| 13 | D | ì | ¥ |
| 14 | E | Ä | ₧ |
| 15 | F | Å | ƒ |

그대로 보내면 알고 있는 위치에 맞춘다. 그런데 이 위치는 프린터가 세팅되어 있는 상태나, 제어코드에 의해 프린트모드의 변함이 일정하지 않아서 리스트를 인쇄할 때 좌우 정렬이 흔들리게 된다.

그래서 탭코드가 들어 오면 저장할 때와 역으로 같은 갯수의 스페이스로 바꿔서 출력하게 된다. 즉 탭코드가 들어오면 8배수의 컬럼을 스페이스 코드로 채운다.

또한 세운7비트 완성형 한글카드에서 HSKEY나 HGKEY를 실행하면 이 프로그램이 프린터로 가는 그래픽캐릭터 앞에 212 코드를 넣어주므로 프린터에서 그래픽캐릭터가 인쇄된다. 이때 프린터는 한글모드로 세팅되어 있어야 한다.

그러나 HSKEY 또는 HGKEY 없이 영문 매뉴얼, 파스칼이나 C 프로그램 소스 리스트 등을 인쇄할 때나, 다른 종류의 한글카드나 아예 한글카드가 없는 컴퓨터에서도 한글없이 영문과 그래픽캐릭터만 있는 화일을 출력하고자 할 때는 /G 옵션으로 깨끗하게 출력시킬 수 있다.

/G 옵션은 그래픽캐릭터 앞에 212 코드를 삽입시켜 준다. 212 코드가 빠지면 그래픽캐릭터를 한글로 인식하여 ([illegible], [illegible], …) 등으로 인쇄되는 사태가 발생한다. 새로 구입한 영문소프트웨어에 있는 도큐멘트화일이나 HELP 화일, README 화일을 인쇄하고자 할 때, 또는 실행중인 프로그램의 화면을 사이드킥으로 떠서 보기좋게 편집한 것을 인쇄할 때 등 그래픽캐릭터가 섞인 화일을 아주 시원하게 인쇄해 줄 것이다.

모든 사용자가 완벽한 환경에 컴퓨터를 다루고자 하지만 대부분 그러한 환경은 희망사항에 그치거나 큰맘(?)먹어야 가능하다. 필자 역시 컴퓨터와 접하는 환경이 좋은 것은 아니다. 그러므로 이것을 극복하려면 좋은 기계로 바꾸는 것이고, 또다른 방법은 현재 가지고 있는 기계를 최대한 이용하는 것이다.

필자는 세운 7비트 완성형 한글카드에서 영문 편집기(Personal Editer)의 빠르고 유용한 기능을 이용해서 한글과 영문, 그래픽캐릭터로 문자를 작성하고 TPRINT로 인쇄한다. 즉, 필자는 위와 같이 간이 워드프로세서를 쓰고 있는데 꼭 전문 워드프로세서가 없어도 고품위의 문서를 작성할 수 있으니 독자 여러분들도 활용해보기 바란다.

## 프로그램 구성

1행～22행 : 프로그램 변수, 상수 선언
23행 : ASCII 값을 구하는 함수
26행 : 문자열이 포함되어 있는지 검사하는 함수
29행 : 정수를 문자열로 바꾸는 함수
33행 : 화일이 존재하는지 검사하는 함수
42행 : 사용방법을 보여주는 루틴
56행～86행 : 옵션을 판별하는 루틴
87행～112행 : 프린트할 것인지 스킵할 것인지를 묻는 루틴
113행 : 다음 페이지로 넘어가게 함
116행～145행 : 화일에서 한 라인을 읽어들임
119행～124행 : 화일에서 한 글자를 읽어들임
146행～163행 : 탭코드를 스페이스로 바꿈
164행～179행 : 한 페이지를 넘김
180행～241행 : 한 페이지를 프린트함
183행～188행 : 화일이름과 페이지를 씀
189행～214행 : 그래픽코드와 일반문자의 크기를 조절함
230행 : LX 프린터일 때 코드변환
243행～284행 : 이 프로그램의 메인 루틴
256행～265행 : 옵션과 화일이름을 판별
269행～272행 : 화일 크기를 얻음
276행～281행 : 페이지 단위로 프린트 스킵을 결정함

〔리스트 1〕 TPRINT 소스 프로그램

```
Program Tprint ;  (********* For text file print to Epson printer *********)
uses
  Dos,Printer,Crt     ;
const
   ESC     = chr(27)  ;
   CReturn = chr(13)  ; LFeed  = chr(10) ;
   Eject   = chr(12)  ;
   Small   = chr(15)  ; Normal = chr(18) ;
   Large   = ESC+'D'  ; NLarge = ESC+'d' ;
   PREGRC  = chr(212) ; { Epson-printer Graphic definition code }
   Author  = '  made by Yang-Wang-Sung. ';
type
   filespec = string        ;
var
   Sourcefile : text         ;
   Sourcename : filespec     ;
   Forfilsize : file of byte ;
   header,Smode,p136,Engmode,Qcount,PError ,Skipage,Continue : boolean ;
   printerc,prnset,Temstr,Prnstr  : string  ;
   Fsize,Linenum,Curpoint         : longint ;
   Rmargin,Pagenum,Prow,i,Maxline : integer ;
Function asc( Temchr:char ) : integer ;
   Begin asc:=Ord(Temchr)
   End ;
Function bos( search,boss:string ) : Boolean ;
   Begin bos:=( pos(search,boss) > 0 )
  . End ;
Function Int2Str( I:integer ; W:integer ) : string ;
   var S:string[11] ;
   Begin str(I:W,S) ; Int2Str:=S
   End ;
Function Existfile(var Sourcename:Filespec) : Boolean ;
   var reg : Registers ;
   Begin reg.ah := $4e ;
         reg.ds := seg(Sourcename)    ;
         reg.dx := ofs(Sourcename)+1 ;
         reg.cx := 0 ;
         msdos(reg)  ;
         Existfile := ((reg.flags and 1)=0 )
   End ;
Procedure Usage ;
   Begin
      writeln(' Usage : TPRINT  filename  [ /n /c /132 /LX /s /g ] ') ;
      writeln('                           ----------------------Option ');
      writeln('                        /n ... Header not print.   ') ;
      writeln('       ** Default **    /c ... line-number Count.  ') ;
      writeln('       * 80 column *    /132.. 132 column printer. ') ;
      writeln('       * LQ-24 pin *    /LX... LX-..9 pin printer. ') ;
      writeln('       *************    /s ... Small-mode printing.') ;
      writeln('                        /g ... English Graphic-mode') ;
      writeln(' first column control  . <Dot>-- then not printing line.');
      writeln('                       .new   -- Eject for new page.');
      Halt(0)
   End ;
Procedure Getoption ;
   var Option,Tstr : string ;
      Temchr      : char   ;
      i,j         : integer;
   Begin
      If copy(Paramstr(2),1,1)<>'/' then Usage ;
      Option := '' ;
      For i:=2 to Paramcount do Begin
         Tstr := Paramstr(i) ;
         for j:=1 to length(Tstr) do
            If bos(UpCase(Tstr[j]),'/NLX1326SGC')
               then Option :=Option + UpCase(Tstr[j])
               else Usage
      End { for i } ;
      If bos('N',Option)  then Header :=False;
      If bos('S',Option)  then Smode  :=True ;
      If bos('C',Option)  then Qcount :=True ;
      If bos('G',Option)  then Engmode:=True ;
      If bos('132',Option) or bos( '136',Option)
                          then p136   :=True ;
      If bos('LX',Option) then Begin Printerc :='LX' ;
                                     Smode    :=True ;
                                     Rmargin  :=80   ;
                               End ;
      If Smode then Prnset:=Prnset+small ;
      If (P136 and (Printerc='LQ')) then If Smode then Rmargin:=255
                                                  else Rmargin:=136
```

```
                                        else If Smode then Rmargin:=192
                                                          else Rmargin:=96 ;
      If Qcount then Rmargin:=Rmargin-6
   End ;
Function QSkipage(pagenum:integer ) : boolean ;
   var Temchr : char    ;
   Begin
      writeln(' 1.Printing-page  2.Skip-page  3.Continue-print  0.Quit');
      write('[',Pagenum:3,' Page ] Select > ') ;
      Temchr:=' ' ;
      If Continue and ( not Keypressed )
         then Temchr:='C'
         else While not bos(Temchr,'1P2S3C0Q') do
                     Temchr:=UpCase(readkey) ;
      Case Temchr of
         '1','P' : Begin writeln(' Printing. . . ') ;
                          QSkipage:=false
                    End ;
         '2','S' : Begin writeln(' Skip',Pagenum:2,' page  ') ;
                          QSkipage:=True  ;
                          Continue:=false
                    End ;
         '3','C' : Begin writeln(' Continue printing until <ESC>.');
                          QSkipage:=false ;
                          Continue:=True
                    End ;
         '0','Q' : halt(0)
      End { Case } ;
      writeln
   End ;
Procedure newpage ;
   Begin Prow:=Maxline
   End ;
Procedure Lineread(var Temstr:string ; var i:integer ) ;
   var Temchr  : char    ;
       Temint  : integer ;
   Function sou2char (var Curpoint:longint) : char ;
      var Tchr : char    ;
      Begin read(Sourcefile,Tchr) ;
            sou2char:=Tchr ;
            inc(Curpoint)
      End ;
    Procedure Append(var Temstr:string ; var i:integer ; Tchr:char ) ;
       Begin Temstr[i]:=Tchr ;
             inc(i)
      End ;
   Begin
      i:=1   ; Temstr := ''  ; Temchr:=' ' ;
      While (i<255) and (Temchr<>LFeed) and (Curpoint<Fsize) do Begin
         Temchr :=sou2char(Curpoint)    ;
         If (Temchr=ESC) then Begin
            Temchr :=sou2char(Curpoint) ;
            If (Temchr<>' ') and (Temchr < chr(127))
               then Append(Temstr,i,ESC)          { control code ESC }
               else Append(Temstr,i,chr(27+128)) ; { Grapic code arrow }
         End { If } ;
         Temint :=asc(Temchr) ;            { controling HT , LF , FF , CR }
         If ((Temint<32) and ( Temint>13 )) or (Temint<9)
            then Temchr := chr( Temint+128 ) ; { Not control->Graphiccode }
         Append(Temstr,i,Temchr)
      End { While in line } ;
      inc(Linenum) ;
   End ;
Procedure Tabcontrol(var Temstr:string ; var ii:integer ) ;
   var Tems1 : string ;
       Temc  : char    ;
       ic,i,it: integer;
   Begin
      it:=1 ; Tems1:='' ; ic:=0 ;
      For i:=1 to ii do Begin
         Temc:=Temstr[i] ;
         If asc(Temc)=9
            then repeat Tems1:=Tems1+' ' ; inc(ic) ;inc(it)
                     until (trunc(ic/8)*8=ic)
            else Begin Tems1:=Tems1+Temc ; inc(ic) ;inc(it)
                     End ;
         If it>=9 then it:=1
      End { for } ;
      Temstr:=Tems1 ; ii:=ic ;
      inc(Prow)     ; If (Rmargin<ic) then inc(Prow)
   End ;
Procedure Nextpage ;
```

```
  var Temchr   : char    ;
      i,Temcnt : integer ;
  Begin
    Prow := 1 ;
    If header then inc(Prow) ;
    While ((Prow<Maxline) and (Curpoint<Fsize)) do Begin
      Lineread(Temstr,i) ;
      Tabcontrol(Temstr,i) ;
      Case Temstr[1] of
        '.'    : If ( bos('new',Temstr) or bos('NEW',Temstr))
                    then Newpage ;
        chr(12): Newpage
      End { case }
    End { While of page 60 line } ;
  End ;
Procedure Printpage ;
  var Temchr   : char    ;
      i,Temcnt : integer ;
  Procedure Pheader( filename:string ; Pagenum:integer ) ;
    Begin inc(Prow) ;
        if Smode then write(lst,small) ;
        write( lst, '+ . .' , 'File : '+filename:(Rmargin-17)) ;
        writeln(lst,' / Page ',Pagenum:3,'*' )
    End ;
  Procedure Encoding(Temcnt:integer ; pasc,pgrc:string ) ;
    var Prncnt,i        : integer ;
        Qchrgr,semigrc : boolean ;
    Procedure Append( Pchr:string ) ;
      Begin Prnstr:=Prnstr+Pchr ;
            inc(Prncnt)
      End ;
    Begin
      Prncnt := 1 ; semigrc:= false ; Prnstr:='' ;
      For i:=1 to Temcnt-2 do Begin
        Temchr := Temstr[i]            ;
        Qchrgr := Temchr > chr(127)    ;
        If not(semigrc=Qchrgr) then Begin
          semigrc := not semigrc       ;
          If Qchrgr then Append(pgrc)
                    else Append(pasc)
        End { If }  ;
        If (Engmode and Qchrgr) then Append(PREGRC) ;
        Append(Temchr)
      End { For of line } ;
      If semigrc then Begin semigrc := false ;
                             Append(Pasc)
                       End ;
      Append(Temstr[i]) ;
      Append(Temstr[i+1])
    End { Encoding } ;
  Begin { Printpage }
    Prow := 1 ;
    If header then Pheader(sourcename,Pagenum) ;
    writeln( lst, Prnset) ;
    While ((Prow<Maxline) and (Curpoint<Fsize)) do Begin
      Lineread(Temstr,i) ;
      Tabcontrol(Temstr,i) ;
      Case Temstr[1] of
        '.'    : If (bos('new',Temstr) or bos('NEW',Temstr))
                    then Newpage ;
        chr(12): Newpage
        else Begin
               Temcnt:=i ; Prnstr:='' ;
               If Printerc='LQ'
                  then Encoding(Temcnt,Normal,Small)
                  else Begin Encoding(Temcnt,Large,NLarge) ;
                             Prnstr:=Large+Prnstr
                       End ;
               If Qcount then
                  Prnstr:=Int2str(Linenum,4)+'] '+Prnstr ;
               If Smode then Prnstr:=Small+Prnstr ;
               write( lst, Prnstr )
             End { else Case }
      End { Case first column }
    End { While of page 60 line } ;
    write( lst, Eject, chr(27)+'@' )   { Putter }
  End { of Procedure Printlist } ;

Begin (* of Program Tprint ********************************************)
  Directvideo:=False ;
  writeln('<*> Text-file Printing to Epson printer.  v0.0507'+Author);
  writeln ;
```

```
Pagenum := 1      ;   Linenum := 0     ;
Curpoint:= 0      ;
Header  := True  ;
Smode   := False ;   Engmode := False ;
Qcount  := False ;
Printerc := 'LQ' ;   P136    := False ;
Maxline := 61     ;   Rmargin := 96     ;
Prnset  :=chr(27)+'h'+chr(27)+'W'+chr(27)+'V' ;
PError  := False ;
If (Paramcount<1)
  then PError:=True
  else Begin
         If Paramcount > 1 then Getoption   ;
         Sourcename := Paramstr(1)+chr($00) ;
         If not(Existfile(Sourcename)) then Begin
           writeln(' Error : File( '+sourcename+' ) is not found !');
           PError:=True
         End { If }
       End { else If } ;
If PError
  then Usage
  else Begin
         assign(Forfilsize,Sourcename) ;
         reset(Forfilsize) ;
         Fsize := filesize(Forfilsize)-1 ;
         close(Forfilsize) ;
         assign(Sourcefile,Sourcename) ;
         reset(Sourcefile) ;
         Continue:=False ;
         While (Curpoint<Fsize) do Begin
           Skipage:=Qskipage(Pagenum) ;
           if Skipage then Nextpage
                      else Printpage ;
           inc(Pagenum)
         End { of File  } ;
         close(Sourcefile)
       End { else If } ;
End. (* of Program Tprint ****************************************************)
```

# NKP에서의 윈도우 구현

구준회/ 고대 컴퓨터클럽

**윈도우 프로그램은 윈도우가 가지는 특성때문에 많은 사람들이 관심을 가져왔다. 그러나 이러한 윈도우 프로그램을 실제로 작성하려면 상당히 힘이 들고 까다롭다. 특히 NKP상에서 NKP의 특징을 이용하여 구성한 윈도우는 찾아보기가 어렵다. 여기에 소개하는 윈도우 라이브러리는 NKP상에서 윈도우를 구현할 수 있게 해주는 좋은 프로그램이다.**

우리가 접할 수 있는 많은 프로그램들 중에 윈도우 프로그램은 윈도우가 가진 특성 때문에 많은 관심의 대상이 되어 왔다. 그러나 지금까지 나왔던 윈도우 중에는 우리가 한글을 쓰기 위하여 많이 사용하는 NKP상에서 NKP의 특징인 그래픽 모드를 이용한 윈도우는 없었다. 이에 필자는 NKP상에서 그래픽 모드의 특성을 최대한 이용한 윈도우를 구현하는 방법을 찾게 되었고 아직은 완전한 기능이 제공되는 것은 아니지만 NKP상에서 윈도우를 구현할 수 있는 터보 C 버전2.0을 이용하여 라이브러리 프로그램을 작성하여 보았다. 완전히 기능은 다음 기회에 추가하기로 하고 윈도우가 구현되는 방법과 그 구조에 관하여 살펴보고 리스트 1을 참고로 독자들의 프로그램에 응용해보기 바란다.

## 스택구조의 멀티윈도우

윈도우를 구현하는데 있어서 몇가지 방법이 있는데 여기서는 복수개의 윈도우를 동시에 열 수 있는 멀티윈도우를 구현했다. 이러한 멀티윈도우를 구현하기 위해서 정보를 저장하는 장소가 필요한데 이것을 스택으로 해결하였다. 스택은 그림 1과 같이 먼저 들어간 원소가 나중에 나오게 되는 자료구조이다. 즉, 삽입과 삭제가 top이라는 한쪽 끝에서 이루어진다.

이러한 스택의 특징 때문에 윈도우들은 맨 마지막으로 만들어진 윈도우만이 다른 윈도우 조작함수들의 영향을 받게 된다. 리스트 1에도 스택 조작 함수는 wpop과 wpush가 있는데 이들은 윈도우의 정보를 다음의 형식으로 저장한다.

```
struct winfo {
  int num,x,y,xl,yl;
  /* x,y : 윈도우의 시작점.  xl,yl :윈도우의 가로세로 길이 */
  char *ssptr;
  /* 윈도우가 생기기전에 있던 화면 내용을 저장한 메모리 포인터 */
  struct winfo *next
  /* 다음 정보의 포인터 */
};
```

## 윈도우 조작 함수

윈도우 조작함수는 closewindow(), openwindow()이고 이 함수들에 포함되는 즉, 윈도우를 생성시키면서 윈도우가 있던 자리의 내용을 저장하는 함수나 재생시키는 함수, 윈도우의 정보를 보관하는 스택 조작함수 등이 있다. 각 함수들의 기능과 특징을 설명하면 다음과 같다.

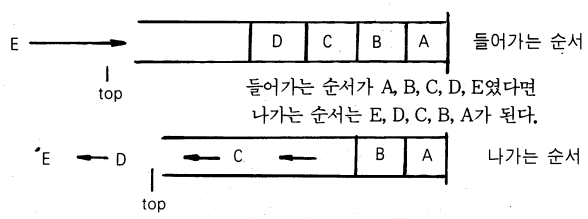
**그래픽 함수들**

dp() : x, y 좌표에 점을 찍는다.
drawHline() : 수평선을 긋는다.
drawVline() : 수직선을 긋는다.
clswindow() : 윈도우가 생기는 곳을 깨끗하게 지운다.
drawbox() : 윈도우의 테두리선을 그린다.

**윈도우 조작 함수**

savescreen() : 윈도우가 생길 화면의 내용을 저장한다.
recoverscreen() : 윈도우가 생겼던 화면의 내용을 복구한다.
closewindow() : 윈도우를 닫는다.
openwindow() : 윈도우를 연다.

〔그림 1〕 스택의 구조



여기서 몇가지 중요한 함수들은 동적인 메모리 관리를 사용하였는데 이 동적인 메모리는 조금 뒤에 언급하겠지만 메모리를 효율적으로 관리한다.

## 허큘리스 카드의 번지와 실제 어드레스

허큘리스 카드는 0과 1 두개의 그래픽 페이지를 가지고 있는데 그 번지는 B800 : 0000번지와 B000 : 0000 번지이다. 그중에서 NKP는 그래픽 번지 0을 사용하고 640×400의 해상도를 갖는다. 그리하여 그래픽 상태를 마치 텍스트 상태인냥 사용하고 있는데, 여기서도 텍스트 상태의 좌표를 입력받도록 되어 있다. 그렇기 때문에 다음과 같이 텍스트 상태의 좌표를 그래픽의 좌표로 바꾸어 주는 계산이 필요하다.

**텍스트 좌표×16=그래픽 좌표의 논리적 어드레스**

(✱ 여기서 16은 각 글자(즉 폰트)들을 이루는 픽셀들의 갯수이다.)

그다음 이러한 논리적인 어드레스를 실제적인 허큘리스 카드의 어드레스로 바꾸어 주어야 하며 그 계산 방법은 다음과 같다.

**임시 논리어드레스=논리어드레스 세로 mod 4**
**실제어드레스=(0×2000×임시 논리어드레스)+**
**(80×(임시 논리어드레스/4))+논리어드레스 가로**

이렇게 계산된 어드레스를 이용하여 다음과 같이 화면을 지우는 함수를 만들 수 있다.

```
int clswindow(int sx, int sy, int ex, int ey)
{
  ....
  sy = sy * 16;
  ey = ey * 16;  /* 논리적 번지 계산 */
  tx = sx;
  for(;sy<ey;sy++) {
     for(sx=tx;sx<ex;sx++) {
     ty = sy % 4;  /* mod operation */
     index = (0x2000*ty)+(80*(sy/4))+sx;  /* 실제번지 계산 */
     *(ptr+index)=0x00;
  }
 ....
 }
```

## 포인터

무엇보다도 프로그램을 작성하다보면 메모리 관리가 문제점으로 대두되고 있는데 이 라이브러리 프로그램도 그래픽 메모리가 차지하는 메모리로 인해 효율적인 메모리 관리를 위하여 포인터를 도입하게 되었다. 대체적으로 C 언어를 공부하다 보면 사람들이 가장 많이 고생하는 부분이 포인터라 할 수 있는데 C 언어에서 사용되는 포인터의 개념은 파스칼과 거의 유사한 개념이다. 이해를 돕기 위해 어셈블리가 가지고 있는 어드레스라는 개념을 설명해 보겠다.

어드레스란 컴퓨터의 메모리 내의 실제적인 번지가 된다. 컴퓨터의 메모리는 1바이트(8비트)의 단위로 번지들이 주어져 있고, 어떠한 연산이나 조작을 할 때면 이러한 번지에 들어있는 내용을 가져오게 되는데 이것은 마치 집의 주소와 그 집에 살고 있는 사람들에 비유된다. 좀더 자세한 이해를 돕기 위해 **그림 2**를 살펴보자.

그림에서 A라는 학교로 철수를 데려오기 위해서는 서울에서, 민수는 대구에서 데려와야 한다. 말을 바꾸어 ( )에 써놓은 말들로 바꾸면 53이란 값을 CPU로 가져오기 위해서는 0000번지의 내용을 가져와야 하고 48이란 값을 가져오기 위해서는 0003 번지의 내용을 가져와야 한다. 여기에서 집주소가 바로 포인터가 되는 것이다. 즉, 어떤 내용이 들어있는 장소를 가르쳐 주고 있는 것이 된다.

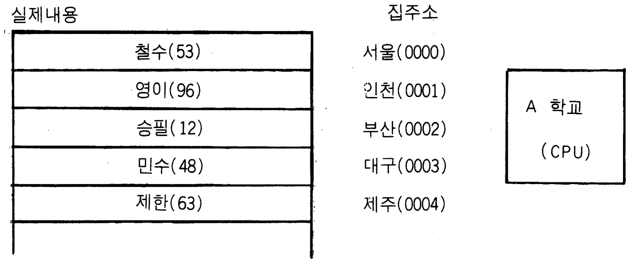
리스트 1에서는 이러한 포인터를 인덱스라는 보조적인 포인터와 같이 사용하여 허큘리스 카드의 그래픽 메모리를 관리한다. 다음은 dp( )라는 함수를 이용한 예이다.

```
void dp(int x, int y)
{
  ....
  unsigned char far *ptr = (char far*) 0xB0000000;
  /* 허큘리스 카드의 0번 페이지의 시작 번지를 가지는 포인터 선언 */
  ....
  /* index 포인터를 찾기위한 연산이 있다.  */
  ....
  bit_mast = *(ptr + index);
  /* 허큘리스 카드의 시작 번지에 인덱스 값을 더해서 필요한 허큘리스 */
  /* 카드의 내용을 가져온다.  */
  ....
  *(ptr + index) = bit_mastk;
  /* 허큘리스 카드에 원하는 값을 넣는다.  */
}
```

## 동적인 메모리관리

동적인 메모리관리(Dynamic memory allocation)는 다른

(그림 2)

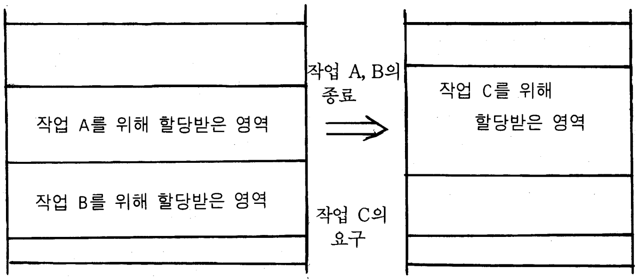


말로 동적인 메모리 할당으로 표현되기도 한다. 동적이란 말이 암시하고 있는 것처럼 할당된 메모리가 가만히 있는 것이 아니라 나타났다 없어졌다 하는 것이다. 물론, 그렇다고 기억소자로 사용되는 램이 나타났다 없어졌다 하는 것이 아니라 그림 3과 같이 프로그램이 어떤 일을 하기 위하여 메모리가 필요하면 그 만큼의 메모리를 컴퓨터에게서 할당받는 것이다. 그리고 그 할당받은 메모리는 일이 끝난 다음에는 다시 컴퓨터(사실은 OS)에게 돌려준다. 이러한 동적인 메모리 관리의 장점은 필요없는 메모리를 보관해 두었다가 다른 필요한 곳에 사용할 수가 있기 때문에 효율적인 메모리 관리를 할 수 있다.

파스칼에서는 동적인 메모리를 할당받기 위해서 new()란 함수를 사용하는데 C에서는 malloc(memory allocation)과 같은 함수를 사용하게 된다. 메모리관리에 있어서 중요한 것은 메모리 모델과 포인터의 지정이다. 만약, 이것이 제대로 일치하지 않으면 프로그램은 십중팔구 실행되지 않고 다운이 되버린다. (실은 필자가 C를 다루기 시작했을때 이런 실수를 많이 했다.) 라이브러리의 openwindow()와 closewindow()에서 윈도우가 생기는 곳의 내용을 저장하거나 혹은 다시 복구하기 위해 다음과 같이 동적인 메모리 관리가 사용되고 있다.

```
int openwindow(int winum, int sx, int sy, int xl, int yl, int box)
{
   ....
   char *tempssptr;  /* 포인터 선언 */
   ....
   tempssptr = (char *)malloc(sizeof(char)*xl*yl*32);
  /* 도스로 부터 화면을 저장하기 위한 메모리를 할당받는다.  */
   ....
   if (tempssprt==NULL) { prnitf(" ....  "); return(0); }
  /* tempssptr 의 내용이 NULL이면 도스로 부터 필요한 메모리를 */
  /* 할당받지 못한 것으로 에러메시지를 출력하고 리턴한다.  */
   ....
}

int closewindow()
```



〔그림 3〕 메모리

```
{
   ...
   free(tempwi->ssptr);  /* 화면을 저장하기 위하여 할당받은 */
                         /* 메모리를 도스로  되돌려 준다.  */
   free(tempwi);  /* 윈도우의 정보를 저장하는 데 사용했던 */
                  /* 스택의 메모리를 도스로 되돌려준다.  */
}
```

## 메모리모델과 Far Pointer

C언어는 각 컴파일러마다 다른 메모리 형태를 구성하고 있다. 대략 5~6가지가 있는데 자세한 사항에 관하여는 사용하려는 컴파일러의 유저가이드나 리퍼런스 가이드를 참고하기 바라고 여기서는 터보 C에 대하여 알아보자.

터보 C에는 Tiny, Small, Medium, Compact, Large, Huge 등의 여섯가지 메모리 모델을 가지고 있다. 이 메모리 모델들을 사용하려면 특수한 포인터인 far pointer가 필요하다. 보통 메모리는 세그먼트와 오프셋으로 나누고 있으며 64K가 하나의 세그먼트를 이루고 있다. 만약 1M 정도의 메모리를 사용할 경우는 많은 세그먼트로 나누어야 하는데 이때 각각의 세그먼트들에 대한 정보를 far pointer가 가지고 있어 메모리를 관리할 수 있는 것이다.

far pointer와 비교되는 것으로 near pointer가 있는데 이것은 단지 오프셋 정보만을 가지고 있다. 메모리모델은 far pointer의 사용 유무에 따라 두가지로 나누고 있는데 far pointer를 사용해야 하는 것은 Compact, Large, Huge이고 나머지 Tiny, Small, Medium은 사용하지 않아도 된다. 여기서는 Large와 Small을 가지고 설명하겠다.

Small 모델은 일반적인 프로그램들에 적합한 형태로 64K의 코드와 64K의 정적인 데이터 영역을 가지게 된다. Large 모델은 64K의 코드를 가지는 점은 같으나 1M까지의 데이터 영역을 가질 수 있다. (far pointer가 필요한 이유는 바로 여기에 있다.)

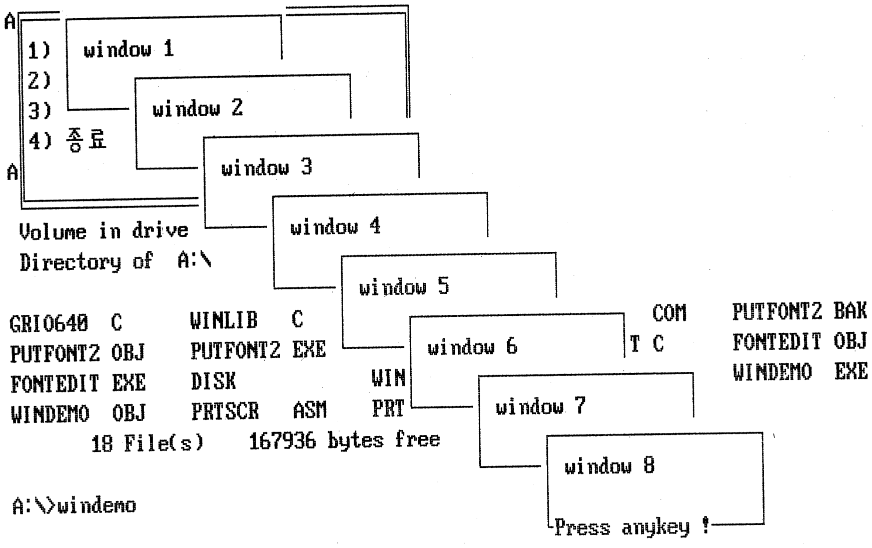
다음 예를 보면 far pointer를 사용했을 경우 이 포인터에는 세그먼트와 오프셋의 정보가 들어갔음을 알 수 있다.

```
      char far *ptr ;  /* far pointer */
  ptr에 들어있는 내용 4F56:6434 (<- 컴퓨터의 상태에 따라 다르다.)
                 segment : offset

near pointer를 사용했을 경우
          char *ptr; /* near pointer */
       ptr에 들어있는 내용    DS:6434
          (* DS 는 데이타 세그먼트 레지스터)
```

여기의 윈도우 라이브러리는 Small모델을 표준으로 만들었기 때문에 이러한 내용이 들어있지는 않지만 이 라이브러리를 보다 큰 메모리 모델을 사용하는 프로그램에서 이용하

〔화면 1〕 윈도우 라이브러리를 이용한 데모프로그램




려면 malloc()와 free() 함수들을 far pointer용인 farmalloc()와 farfree()로 바꾸어 주어야 한다. 또, 사용된 포인터들도 far 포인터로 바꾸어 준다.

## 프로그램의 응용

필자는 이 라이브러리의 손쉬운 활용을 위해 화면 1과 같은 간단한 데모프로그램을 작성해 보았다. 이 라이브러리는 풀다운 방식의 메뉴선택 기능을 갖추고 있는데 이 기능은 다음 세가지 함수로 구성이 된다.

changeatt() : 문자의 속성을 바꾸어준다.
getarrow() : 화살표키의 입력을 받아서 적절한 값을 돌려준다.
selectmenu() : 현재 활성화된 윈도우(마지막으로 열린)에서 메뉴를 선택하게 해준다.

데모 프로그램을 실행시켜 보면 이 라이브러리가 가진 특성과 기능이 다음과 같다는 것을 쉽게 알 수 있다.

① 멀티윈도우

이 윈도우 라이브러리는 동적인 메모리 할당을 사용했기 때문에 메모리가 허용하는 한도에서 몇개고 윈도우를 동시에 열 수 있는 복수개의 윈도우를 제공한다.

② 윈도우 개폐속도의 표시

윈도우를 열고 닫을 때 걸리는 시간을 알려 주는데 그래픽상에서 이루어지는 처리이기 때문에 텍스트에서 하는 화면처리보다 느리다. 하지만 적당한 크기의 윈도우를 사용한다면 화상처리가 느린 것은 어느 정도 극복할 수 있다.

③ 메뉴선택

데모의 전체가 풀다운 방식의 메뉴를 선택하도록 하였는데 별다른 내용은 없고 단지 메시지 만을 출력하게 한다.

## 맺음말

여기까지 설명한 윈도우 라이브러리는 앞에서 설명한 대로 완전한 형태의 윈도우 처리 라이브러리는 아니다. 때문에 열어 놓은 윈도우 중에서 어떤 것이라도 임의로 선택할 수 있는 기능과 윈도우내에서 스크롤, 보다 완전한 그래픽 유저 인터페이스 등이 보강되어야 한다. 또한 이 라이브러리를 컴파일하다 보면 컴파일시의 경고가 하나 발생하는데 이 경고는 필자가 차후에 이 라이브러리의 화일을 위해서 만들어 놓은 것이므로 무시해도 된다.

요즘, 좋은 C 컴파일러가 많이 나오면서 좋은 기능을 가진 함수 라이브러리들이 많이 제공되고 있다. (사실 필자가 처음 터보 C버전 1.0을 사용했을 때는 그래픽 라이브러리는 커녕 gotoxy()라는 함수도 필자가 만들어 썼다.) 하지만 이러한 라이브러리들의 소스는 구할 수가 없을 뿐더러, 리퍼런스 가이드에 나와있는 내용도 이러한 라이브러리를 충분히 설명해주지 못하는 경우가 많다. 그 때문에 필자는 가끔 필요한 라이브러리를 만들어 쓴다. 또, 터보 파스칼에 보면 툴박스라는 것들이 있는데, 아직까지 C용으로 이러한 툴박스들이 흔하지 않다. C언어는 특징상 언어자체에 제공되는 가능이 많지 않다. 그 때문에 풍부한 라이브러리가 필요하다. 앞으로 많은 C 언어 사용자들이 나와서 편리한 라이브러리들이 많이 만들어졌으면 하는 바램이다.

〔리스트 1〕 윈도우 라이브러리 프로그램

```
/* Window Library for NKP (640 X 400 Herc.)  ver 1.0 */
/* Program by Ku Jun-whoe                            */
/* Korea Univ. Computer Club (K.U.C.C.)              */
/* from 1989 - 4 - 19  to                            */
/* Using Turbo C ( memory model small )              */
/* notice : this library need two header file ( stdio.h
                                        and alloc.h) */
struct winfo {
  int num,x,y,xl,yl;
  char *ssptr;         /* saved screen pointer */
  struct winfo *next;
};

struct winfo *wtop;     /* global variable : stack top */
int c_windownumber ;    /* global variable : current window
                                                  number */
/* ----------------- graphic operation --------------- */
/* dp(x,y) is only draw a point in white */
void dp(int x,int y)
{
  unsigned char bit_mask,co=1;
  int index,ty;
  unsigned char far *ptr = (char far*) 0xB0000000;
  /* expected memory position of  GraphicMode   */

  ty = y%4;
  index=(0x2000*ty)+(80*(y/4))+(x/8);
  ty = 7-(x%8);
  bit_mask = *(ptr + index);
  co = co<<ty;     /* shift left   */
  bit_mask = bit_mask|co;
  *(ptr + index) = bit_mask;
}/* draw point */

/* draw Horizontal line */
void drawHline(int sx, int ex, int sy)
{
  ex++;
  for(;sx<ex;sx++) dp(sx,sy);
}

/* draw Vertical line */
void drawVline(int sy, int ey, int sx)
{
  ey++;
  for(;sy<ey;sy++) dp(sx,sy);
}

/* clear menu window */
int clswindow(int sx, int sy, int ex, int ey)
{
  int index,count,ty,tx;
  unsigned char far *ptr = (char far*) 0xB0000000;

  count=0;
  sx--;sy--;ex--;ey--;
  if ((sx<0)||(sy<0)||(ex>79)||(ey>24)) return -1;
  /* check both sx,sy and ex,ey are regular range */
  sy = sy * 16;
  ey = ey * 16;
  tx=sx;
  for(;sy<ey;sy++){
    for(sx=tx;sx<ex;sx++) {
      ty = sy%4;
      index=(0x2000*ty)+(80*(sy/4))+sx;   /* calculate graphic
      *(ptr+index)=0x00;                               address */
      count++;
    }
  }
  return 1;
}

/* draw box with single or double line */
void drawbox(int sx, int sy, int ex, int ey, int box)
{
  sx--;sy--;ex--;ey--;
  if ((sx<0)||(sy<0)||(ex>79)||(ey>24)) return ;
  switch(box) {
    case 1 : /* draw single line box */
      sx=sx*8+4;  ex=ex*8-4;
      sy=sy*16+8; ey=ey*16-8;  /* calculate x,y position */
      drawHline(sx,ex,sy); drawHline(sx,ex,ey);
      drawVline(sy,ey,sx); drawVline(sy,ey,ex);
    break;
    case 2 : /* draw double line box */
      sx=sx*8+2; ex=ex*8-2;
      sy=sy*16+4; ey=ey*16-4;
      drawHline(sx,ex,sy); drawHline(sx,ex,ey);
      drawVline(sy,ey,sx); drawVline(sy,ey,ex);
      sx=sx+2; ex=ex-2;
      sy=sy+4; ey=ey-4;
      drawHline(sx,ex,sy); drawHline(sx,ex,ey);
      drawVline(sy,ey,sx); drawVline(sy,ey,ex);
    break;
  }
}

/*  ---------- stack operation for menu - window -------------- */

/* push saved screen information */
int wpush (int winum, int sx, int sy, int xl, int yl, char *ptr)
{
```

```
  struct winfo *tempwi;

  tempwi=(struct winfo *)malloc(sizeof(struct winfo));
  /* allocate memory block for window information stack */
  if (tempwi==NULL) { printf(" Memory allocation Error.\n")
                                                    ;return(0);}
  /* check memory allocation error */
  tempwi->num = winum;
  tempwi->x = sx;
  tempwi->y = sy;
  tempwi->xl = xl;
  tempwi->yl = yl;
  tempwi->ssptr = ptr;      /* store information */

  tempwi->next = wtop;
  wtop = tempwi;
  return(1); /* return sucess */
}

/* pop saved screen infomation */
struct winfo *wpop ()
{
  struct winfo *tempwi;
  if (wtop==NULL) {printf("Error: Attempt pop to empty stack.\n")
  tempwi=wtop;                                       ;return(0);}
  wtop=wtop->next;
  return(tempwi);
}

/* --------------  Cusor movement in Window -------------- */
/* gotoxy ( in Boland libray ) is absolute .              */
/* but gotoxywin ( in this libray ) is relative movement  */
void gotoxywin(int x, int y)
{
   gotoxy(x+wtop->x,y+wtop->y);
}

/* --------------  Window Libaray for NKP  ----------------*/

/* save screen */
int savescreen(int sx, int sy, int ex, int ey, char *tempssptr)
{
  int index,count,ty,tx;
  unsigned char far *ptr = (char far*) 0xB0000000;

  count=0;
  sx--;sy--;ex--;ey--;
  if ((sx<0)||(sy<0)||(ex>79)||(ey>24)) return -1;
  /* check both sx,sy and ex,ey are regular range */
  sy = sy * 16;
  ey = ey * 16;
  tx = sx;
  for(;sy<ey;sy++){
    for(sx=tx;sx<ex;sx++) {
      ty = sy%4;
      index=(0x2000*ty)+(80*(sy/4))+sx;   /* calculate graphic
      *(tempssptr+count)=*(ptr+index);               address */
      count++;
    }
  }
}

/* recover screen */
int recoverscreen(int sx, int sy, int ex, int ey, char *tempssptr)
{
  int index,count,ty,tx;
  unsigned char far *ptr = (char far*) 0xB0000000;

  count=0;
  sx--;sy--;ex--;ey--;
  if ((sx<0)||(sy<0)||(ex>79)||(ey>24)) return -1;
  /* check both sx,sy and ex,ey are regular range */
  sy = sy * 16;
  ey = ey * 16;
  tx = sx;
  for(;sy<ey;sy++){
    for(sx=tx;sx<ex;sx++) {
      ty = sy%4;
      index=(0x2000*ty)+(80*(sy/4))+sx;   /* calculate graphic
      *(ptr+index)=*(tempssptr+count);               address */
      count++;
    }
  }
}

/* open  window        sx : start row of window */
/*                     sy : start col of window */
/*                     xl : x length            */
/*                     yl : y length            */
/*                     winum : window number to open */
int openwindow(int winum, int sx, int sy, int xl, int yl, int box)
{
  struct winfo tempinfo;
  char *tempssptr;       /* temp. save scrren ptr */
  int ex,ey;

  ex=sx+xl;
  ey=sy+yl;
  tempssptr = (char *)malloc(sizeof(char)*xl*yl*32);
  /* allocate memory block for save screen */
  wpush(winum,sx,sy,xl,yl,tempssptr);
  /* push saved screen info. */
  if (tempssptr==NULL) { printf(" Memory allocation error.\n")

  savescreen(sx,sy,ex,ey,tempssptr);   ;return(0);}
```

```
  clswindow(sx,sy,ex,ey);         /* clear menu window */
  drawbox(sx,sy,ex,ey,box);  /* draw box with single or double
                                                    line */
  c_windownumber++;           /* increse current window number */
  return (c_windownumber);              /* success */
}

/* close last opened  window */
int closewindow()
{
  struct winfo *tempwi;
  int sx,sy,ex,ey,winum;

  if (c_windownumber==0) { /* no opened window */ return -1 ;}
  tempwi=wpop();
  sx=tempwi->x;
  sy=tempwi->y;
  winum=tempwi->num;
  ex=sx+tempwi->xl;
  ey=sy+tempwi->yl;
  recoverscreen(sx,sy,ex,ey,tempwi->ssptr);
  free(tempwi->ssptr);    /* free saved screen memory block */
  free(tempwi);      /* free window information stack block */
  c_windownumber--;        /* decrease current window number */
  return(c_windownumber);
}
/* ------------------- END : Window library for NKP ------- */

/* This function is tip function for selecting operation. */
/* Only active window effect on this function            */
/* Return value is selected line number.                 */
/* Selectmenu ( main function ? ) need two sub function  */
/* changeatt ( change character attribute ) and          */
/* getarrow  ( get arrow key code ).                     */

/* x, y is start position . */
/* xlength is change length */
void changeatt(int x, int y, int xlength)
{
  int index,tx,ty,ty1,ty2;
  unsigned char temp;
  unsigned char far *ptr = (char far*) 0xB0000000;

  x--;y--;
  y=y*16;
  /* check both sx,sy and ex,ey are regular range */
  tx=x;
  ty=y;
  for(;x<tx+xlength;x++) {
    for(ty=0;ty<16;ty++) {
      ty1 = y + ty;
      ty2 = ty1%4;
      index=(0x2000*ty2)+(80*(ty1/4))+x;
      temp=*(ptr+index);
      *(ptr+index)=temp^0xff; /* bitwise XOR */
    }
  }
}

/* return value 7 is Home    8 is up      9 is PgUp  */
/*              4 is left                 6 is right */
/*              1 is End     2 is down    3 is PgDn  */
/* if other key pressed, return character */
int getarrow()
{
  char c;

  c=getch();
  if (c!=0) { return c ;} /* other key pressed */
  c=getch();
  switch(c) {
    case 0x4f : return 1;
    case 0x50 : return 2;
    case 0x51 : return 3;
    case 0x4b : return 4;
    case 0x4d : return 6;
    case 0x47 : return 7;
    case 0x48 : return 8;
    case 0x49 : return 9;
  }
}

int selectmenu()
{
   int tn=1,maxitem;
   char c=0;

   maxitem=wtop->yl-2;
   if (wtop==NULL) return(0);   /* none window opened */
   changeatt(wtop->x+1,wtop->y+tn,wtop->xl-2);
   while(c!=0xd) {             /* 0xd is CR's code */
     c=getarrow();
     changeatt(wtop->x+1,wtop->y+tn,wtop->xl-2);
     switch(c) {
       case 2 : tn++; if (tn>maxitem)  tn=1; break;
       case 8 : tn--; if (tn<1) tn=maxitem; break;
     }
     changeatt(wtop->x+1,wtop->y+tn,wtop->xl-2);
   }
   changeatt(wtop->x+1,wtop->y+tn,wtop->xl-2);
   return(tn);
}
```

# 램디스크

박승제/여주컴퓨터학원장

컴퓨터를 사용하다보면 디스크에 입출력하는 시간이 너무 길어 짜증이 나는 경우가 있다. 물론 여유가 있다면야 하드디스크 드라이브를 달면 해결이 되겠지만 주머니가 빈약한 사람에게 하드디스크 드라이브는 그림의 떡에 불과하다. 좋다는 것은 알고 있지만 어디 하드를 단다는 것이 디스켓 한 장 사는 것처럼 쉬운 일인가.

그래도 8비트시절에 카세트를 상대로 로드나 세이브하느라 몇 십분씩 소비해 본 경험이 있는 사람들에게는 현재의 디스크 드라이브의 입출력 속도가 번갯불처럼 빠르게 여겨질지도 모른다. 그러나 개구리 올챙이 시절 모른다고, 플로피를 상대로 하는 작업에 익숙하다 보면 번갯불처럼 빠른 속도가 이제는 굼뱅이처럼 느리게만 느껴진다.

이럴 때는 또다시 어떻게 하면 속도를 빨리할 수 없을까라는 생각이 들 것이다. 이런 상황을 해결해 줄수 있는 것이 바로 램디스크 드라이브이다. 램디스크 드라이브는 입출력속도가 플로피디스크 드라이브에 비하면 최소한 200배쯤 빠르며, 하드디스크 드라이브 보다도 수십배 빠르다. 또한 이것을 설치하는데 단 한푼의 돈도 들지 않는다는 점이 독자들에게 기쁨을 줄 것이다.

램디스크 드라이브는 말 그대로 램의 일부분을 떼어내서 드라이브처럼 운용하는 것을 말한다. 예전에는 메모리의 값이 비싼 까닭에 컴퓨터의 메모리가 충분치 못했다. 따라서 작은 메모리에서 큰 프로그램을 운용하기 위해서 디스크 드라이브를 컴퓨터의 메모리처럼 운용하는 가상메모리(Vertual Memory)라는 기법을 사용해 왔다. 그러나 지금은 메모리의 값이 상대적으로 저렴해져 요즘 PC들은 사용자가 다 사용하지 못할 정도로 많은 메모리용량을 탑재하고 있으며, 가상메모리와는 반대되는 개념으로 가상드라이브, 즉 램디스크 드라이브가 등장한 것이다.

램디스크 드라이브는 램에 화일을 저장한 까닭에 컴퓨터의 전원이 꺼지면 그 내용도 자동적으로 지워지게 된다. 따라서 플로피나 하드의 경우처럼 포맷할 필요도 없으며 따로 삭제할 필요도 없다. 또, 램디스크 드라이브는 2개 이상의 드라이브를 필요로 하는 경우에도 아주 유용하게 이용할 수 있다.

이를테면 터보 C의 경우 시스템 디스크를 A:에, 라이브러리를 B:에 넣고 작업을 하게 된다. 그러다보니 사용자의 프로그램을 저장하거나 컴파일할 경우에는 일일이 디스크를 교환하지 않으면 안된다. 이때에 램디스크 드라이브를 사용하면 매우 유용하다. 즉 2대의 플로피를 탑재한 기종에서는 램디스크 드라이브가 C:를 배정받으므로 사용자가 작성한 프로그램이나 컴파일된 오브젝트는 C:로 저장하도록 하면 디스크를 교환할 필요가 전혀 없다. 또한 게임 등을 할 때 디스켓의 모든 화일을 램디스크 드라이브에 복사해 두고 게임을 실행하면 플로피에서 실행하는 경우보다 화면 처리가 훨씬 더 빨라진다.

또 램디스크 드라이브는 DOS의 VDISK.SYS를 이용하는 방법도 있으나 여기에서 소개하려는 'RAMDISK.SYS'를 이용하기를 권하고 싶다. 'VDISK.SYS'는 한번 설치한 후 그 크기를 바꿀 경우에는 'CONFIG.SYS'를 다시 작성한 후 시스템을 리세트시켜야 한다. 그러나 여기에 소개하는 'RAMDISK. SYS'는 컴퓨터가 부팅될 때 화면 1과 같이 램디스크를 얼마의 크기로 설정하겠느냐고 물어오므로 해당 수치를 입력해 주면 된다.

또한 CONFIG.SYS에 여러번 등록해 누면 여러 개의 램디스크를 각각 운용할 수가 있다. 물론 이때 메모리용량이 한계에 부딪치는 경우도 있다. 'RAMDISK.SYS'의 설치는 스스로 부팅할 수 있는 디스켓에 'RAMDISK.SYS'를 복사

〔화면 1〕

```
Columbia Data Products
RAM DISK DRIVER Version 1.0

Choose one of the following sizes:
        1  -   16K Ram disk
        2  -   32K Ram disk
        3  -   64K Ram disk
        4  -  128K Ram disk
        5  -  256K Ram disk
        6  -  384K Ram disk
        7  -  512K Ram disk

Enter choice [1]
```

한 후 다음과 같이 'CONFIG.SYS'를 만든다.

```
COPY CON CONFIG.SYS
DEVICE=RAMDISK.SYS
<Ctrl>+<Z>
```

그 다음에 시스템을 리세트시키면 화면 1과 같은 메시지가 나오게 된다. 이때 램디스크 드라이브의 용량을 너무 크게 잡으면 사용자의 메모리용량이 상대적으로 작아져 수행되지 않는 프로그램도 있을 수 있으므로 적당히 지정해야 한다. 필자의 경우에는 보통 128KB를 지정하는데 게임등을 할 경우에는 디스크 1장 분을 수록할 수 있도록 384KB로 잡는다.

'CONFIG.SYS'의 'DEVICE=RAMDISK.SYS'를 여러번 기술하면 기술한 횟수만큼 램디스크 드라이브를 만들 수 있다. 여러번이라고 해서 무작정 많이 만들 수는 없고, 현재 사용하고 있는 컴퓨터가 가질 수 있는 최대 드라이브수(보통 5개)이내에서 가능하다.

리스트 1은 'RAMDISK.SYS'와 'CONFIG.SYS'를 만들어 주는 베이직 프로그램이다. 이 프로그램을 입력한 후 실행하면 에러가 있는지를 체크한 후 에러가 없으면 시스템을 리세트시키라는 메시지가 나오는데 그때 시스템을 리세트시키면 사용하는 디스크 드라이브에 RAMDISK.SYS와 CONFIG.SYS 화일이 만들어진다. 이때 사용 드라이브가 플로피인 경우는 하드 드라이브에 이 시스템화일을 복사해서 사용해야 한다.

DBASEⅢ 등을 쓰기 위해 'CONFIG.SYS'에 'FILES=15', 'BUFFERS=10' 등을 설정하고 싶으면 480 이후에 다음과 같이 기술하기 바란다.

```
485 PRINT #1, "FILES=15"
487 PRINT #1, "BUFFERS=10"
```

이 프로그램은 먼저 체크섬(Check-Sum;DATA문의 맨 우측 3자리 수치)을 읽어서 이상 유무를 판단한 다음 데이터를 읽어들여 디스크에 수록하게 된다. 체크섬은 데이터를 16개씩 읽어서 합산한 값이므로 실제로 16개씩 읽어서 합산한 값과 체크섬과 비교를 하여 값이 다르면 그 행은 데이터를 잘못 입력한 것으로 간주하게 된다. 그러므로 입력시 데이터의 순서가 바뀐 경우, 이를테면, '40, 30,…'으로 입력 할 것을 '30, 40…'으로 입력한 경우에는 에러 체크를 못하게 된다. 따라서 체크섬을 너무 믿어서도 안된다.

'RAMDISK. SYS'가 독자들의 컴퓨터 생활에 많은 도움이 되길 바란다.

〔리스트 1〕

```
10 '===========================
20 '       RAMDISK.SYS
30 '   - Yeo Ju Computer -
40 '===========================
100 CLS
110 DEFINT A-Z
120 DEFDBL S
130 PRINT "Check Check-Sum ...."
140 FOR I=0 TO 87
150   FOR J=1 TO 16
160     READ A$
170   NEXT J
180   READ A
190   S=S+A
200 NEXT I
210 IF S=116320 THEN GOTO 260
220 PRINT "Check Sum Error !"
230 PRINT "Please Check your Check Sum"
240 END
250 ' ---- Make File ----
260 OPEN "A:RAMDISK.SYS" AS #1 LEN=1
270 FIELD #1, 1 AS F$
280 RESTORE
290 LOCATE 1,1 : PRINT "Loading Data .... ";
300 FOR I=0 TO 87
310   S=0
320   LOCATE 1,19 : PRINT USING"####";I*10+1000
330   FOR J=1 TO 16
340     READ A$
350     A=VAL("&H"+A$)
360     LSET F$=CHR$(A)
370     PUT #1
380     S=S+A
390   NEXT J
400   READ A
410   IF S=A THEN GOTO 440
420   PRINT "Data Error !"
430   END
440 NEXT I
450 CLOSE
460 ' ---- Make Config.sys ----
470 OPEN "A:CONFIG.SYS" FOR OUTPUT AS #1
480   PRINT #1,"DEVICE=RAMDISK.SYS"
490 CLOSE
500 PRINT
510 PRINT "Your System Setup RAM DISK."
520 PRINT "Please Reset Your Computer !"
530 END
1000 DATA FF,FF,00,00,00,00,44,00,4F,00,01,00,00,00,00,00, 658
1010 DATA 00,00,5B,01,7E,00,8B,00,FF,FF,99,00,FF,FF,FF,FF, 2040
```

```
1020 DATA FF,FF,1A,01,54,01,FF,FF,FF,FF,FF,FF,00,00,00,00, 2152
1030 DATA 32,00,00,02,01,00,00,02,40,00,00,00,FA,03,00,00, 372
1040 DATA 00,00,00,00,2E,8C,06,2C,00,2E,89,1E,2E,00,CB,50, 778
1050 DATA 53,51,52,55,56,57,1E,06,16,9C,2E,8E,06,2C,00,2E, 1002
1060 DATA 8B,1E,2E,00,26,8A,47,02,D0,E0,98,8B,F8,2E,FF,A5, 1901
1070 DATA 12,00,9D,17,07,1F,5F,5E,5D,5A,59,5B,58,CB,26,81, 1246
1080 DATA 4F,03,00,01,26,C6,47,0E,01,EB,E7,8C,C8,26,C7,47, 1519
1090 DATA 12,32,00,26,89,47,14,EB,D9,26,8B,47,12,A3,41,00, 1280
1100 DATA E8,1E,00,06,8E,D8,BE,00,00,26,8B,7F,0E,26,8E,47, 1385
1110 DATA 10,E8,2B,00,07,26,C7,47,03,00,01,26,89,4F,12,EB, 1117
1120 DATA B1,26,8B,47,14,3B,06,3A,00,7F,09,B1,05,D3,E0,03, 1324
1130 DATA 06,3F,00,C3,26,C7,47,03,08,10,83,C4,02,EB,93,2E, 1356
1140 DATA 8B,16,41,00,E8,10,00,B9,00,01,F3,A5,4A,75,F5,2E, 1550
1150 DATA 8B,0E,41,00,2B,CA,C3,81,FE,E0,FF,72,0B,8C,D8,05, 2006
1160 DATA E0,0F,81,EE,00,FE,8E,D8,81,FF,E0,FF,72,0B,8C,C0, 2538
1170 DATA 05,E0,0F,81,EF,00,FE,8E,C0,C3,26,8B,47,12,A3,41, 1889
1180 DATA 00,E8,9D,FF,1E,06,26,8B,77,0E,26,8E,5F,10,8E,C0, 1615
1190 DATA BF,00,00,E8,A9,FF,07,1F,26,89,4F,12,80,3E,43,00, 1414
1200 DATA 01,C6,06,43,00,00,75,03,E9,4E,FF,26,C7,47,03,00, 1269
1210 DATA 01,E9,1E,FF,C6,06,43,00,01,EB,BF,CD,12,A3,65,02, 1706
1220 DATA E8,C9,01,8B,F8,B1,02,D3,E7,83,EF,02,8B,85,C1,03, 2282
1230 DATA 8B,D0,E8,F7,00,52,3D,00,10,77,0C,C7,06,38,00,20, 1409
1240 DATA 00,C7,06,3D,00,01,00,B1,05,D3,E8,A3,3A,00,E8,5C, 1437
1250 DATA 01,5A,8C,C9,26,81,4F,03,00,01,26,C6,47,0D,01,26, 1041
1260 DATA C7,47,0E,00,00,26,89,57,10,26,C7,47,12,30,00,26, 974
1270 DATA 89,4F,14,52,50,8D,16,67,02,B4,09,CD,21,58,5A,E9, 1504
1280 DATA B0,FE,40,20,31,32,2D,32,39,2D,38,33,20,38,32,30, 1115
1290 DATA 32,30,2D,30,30,30,31,20,52,41,4D,20,44,69,73,6B, 1019
1300 DATA 20,44,72,69,76,65,72,20,2D,20,56,65,72,73,69,6F, 1393
1310 DATA 6E,20,31,2E,30,30,20,43,6F,70,79,72,69,67,68,74, 1318
1320 DATA 20,28,43,29,20,43,6F,6C,75,6D,62,69,61,20,44,61, 1221
1330 DATA 74,61,20,50,72,6F,64,75,63,74,73,2C,20,49,6E,63, 1455
1340 DATA 2E,20,31,39,38,33,20,41,6C,6C,20,52,69,67,68,74, 1146
1350 DATA 73,20,52,65,73,65,72,76,65,64,2E,20,4C,69,63,65, 1438
1360 DATA 6E,73,65,64,20,4D,61,74,65,72,69,61,6C,20,2D,20, 1382
1370 DATA 50,72,6F,70,65,72,74,79,20,6F,66,20,43,44,50,2C, 1405
1380 DATA 20,49,6E,63,2E,00,00,0A,0D,0A,0D,24,53,0E,5B,B1, 807
1390 DATA 04,D3,E3,81,C3,2C,03,D3,EB,43,89,1E,3F,00,03,D3, 1770
1400 DATA 5B,C3,3D,07,00,74,03,E9,EA,00,8D,16,91,02,E9,E7, 1714
1410 DATA 00,0D,0A,00,2A,2A,2A,20,20,45,52,52,4F,52,20,20, 671
1420 DATA 2A,2A,2A,0D,0A,4D,65,6D,6F,72,79,20,63,6F,6E,74, 1250
1430 DATA 72,6F,6C,20,62,6C,6F,63,6B,73,20,64,65,73,74,72, 1581
1440 DATA 6F,79,65,64,0D,0A,00,43,61,6E,6E,6F,74,20,61,6C, 1304
1450 DATA 6C,6F,63,61,74,65,20,6D,65,6D,6F,72,79,20,66,6F, 1574
1460 DATA 72,20,52,41,4D,20,44,49,53,4B,0D,0A,24,A1,3F,00, 984
1470 DATA 06,8E,C0,8B,16,3D,00,B1,09,D3,E2,89,16,1F,03,A1, 1539
1480 DATA 38,00,B1,04,D3,E0,03,C2,8B,C8,33,C0,33,FF,F3,AB, 2171
1490 DATA BF,00,00,E8,0B,00,8B,3E,1F,03,E8,04,00,07,C3,00, 1107
1500 DATA 00,26,C7,05,FA,FF,26,C6,45,02,FF,C3,50,8D,16,88, 1883
1510 DATA 03,B4,09,CD,21,58,BF,00,00,3B,85,C1,03,7C,09,83, 1361
1520 DATA C7,04,81,FF,1C,00,7C,F1,0B,FF,74,28,8D,16,DD,03, 1789
1530 DATA B4,09,CD,21,D1,EF,D1,EF,8B,CF,8D,16,04,04,B4,09, 2029
1540 DATA CD,21,81,C2,17,00,E2,F6,8D,16,A5,04,B4,09,CD,21, 1815
1550 DATA E8,85,01,C3,8D,16,BA,04,B4,09,CD,21,26,81,4F,03, 1590
1560 DATA 0C,10,83,C4,02,E9,EA,FC,0D,0A,43,6F,6C,75,6D,62, 1709
1570 DATA 69,61,20,44,61,74,61,20,50,72,6F,64,75,63,74,73, 1496
1580 DATA 0D,0A,52,41,4D,20,44,49,53,4B,20,44,52,49,56,45, 988
1590 DATA 52,20,56,65,72,73,69,6F,6E,20,31,2E,30,0D,0A,00, 1054
1600 DATA 24,80,00,00,04,80,00,00,08,C0,00,00,10,00,01,00, 513
1610 DATA 20,80,01,00,40,00,02,00,60,80,02,00,80,0D,0A,43, 671
1620 DATA 68,6F,6F,73,65,20,6F,6E,65,20,6F,66,20,74,68,65, 1494
1630 DATA 20,66,6F,6C,6C,6F,77,69,6E,67,20,73,69,7A,65,73, 1599
1640 DATA 3A,0D,0A,24,09,31,20,20,2D,20,20,20,31,36,4B,20, 590
1650 DATA 52,61,6D,20,64,69,73,6B,0A,0D,24,09,32,20,20,2D, 974
1660 DATA 20,20,20,33,32,4B,20,52,61,6D,20,64,69,73,6B,0A, 1061
1670 DATA 0D,24,09,33,20,20,2D,20,20,20,36,34,4B,20,52,61, 706
1680 DATA 6D,20,64,69,73,6B,0A,0D,24,09,34,20,20,2D,20,20, 861
1690 DATA 31,32,38,4B,20,52,61,6D,20,64,69,73,6B,0A,0D,24, 1068
1700 DATA 09,35,20,20,2D,20,20,32,35,36,4B,20,52,61,6D,20, 819
1710 DATA 64,69,73,6B,0A,0D,24,09,36,20,20,2D,20,20,33,38, 829
1720 DATA 34,4B,20,52,61,6D,20,64,69,73,6B,0A,0D,24,09,37, 1029
1730 DATA 20,20,2D,20,20,35,31,32,4B,20,52,61,6D,20,64,69, 957
1740 DATA 73,6B,0A,0D,24,0A,0D,45,6E,74,65,72,20,63,68,6F, 1160
1750 DATA 69,63,65,20,5B,31,5D,08,08,24,0D,0A,07,2A,2A,2A, 778
1760 DATA 20,20,45,52,52,4F,52,20,20,2A,2A,2A,0D,0A,49,6E, 854
1770 DATA 73,75,66,66,69,63,69,65,6E,74,20,6D,65,6D,6F,72, 1648
1780 DATA 79,20,74,6F,20,69,6E,73,74,61,6C,6C,20,52,61,6D, 1491
1790 DATA 20,44,69,73,6B,0D,0A,24,B8,07,0C,CD,21,3C,0D,75, 1117
1800 DATA 06,B8,01,00,EB,14,90,2C,30,7E,ED,98,3B,C7,7F,E8, 1814
1810 DATA 50,04,30,B4,02,8A,D0,CD,21,58,C3,52,41,4D,20,44, 1505
1820 DATA 49,53,4B,20,44,52,49,56,45,52,20,57,72,69,74,74, 1293
1830 DATA 65,6E,20,62,79,20,45,64,64,79,20,4C,6F,72,2C,20, 1293
1840 DATA 4A,75,6E,65,20,31,39,38,33,20,43,6F,70,79,72,69, 1309
1850 DATA 67,68,74,20,28,43,29,20,43,6F,6C,75,6D,62,69,61, 1347
1860 DATA 20,44,61,74,61,20,50,72,6F,64,75,63,74,73,2C,20, 1370
1870 DATA 49,6E,63,2E,00, , , , , , , , , , , , 328
```

# 터보에디터의 장점을 모두 갖춘 새로운 에디터 BBUN

이기현/서강대 BBUNDEX

'가장 편한 에디터는?' 이런 질문을 받으면 필자는 당연히 터보 에디터(Turbo Editor)라고 말할 것이다. 여지껏 필자가 써본 에디터(게으른 관계로 몇개 못 써봤지만)중에서 가장 편하고 손에 익은 것이 터보 파스칼 에디터이다. 편리한 키배열, 블럭 처리, 깔끔한 메뉴 등 처음에는 파스칼 프로그래밍 때문에 사용하기 시작하였지만 요즈음은 어셈블리어 등을 다룰 때에도 터보 에디터를 사용하고 있다. 심지어는 영문 워드프로세서로서 워드스타 대신에 이 에디터를 사용하기도 하는데 (물론 짧은 내용) 많은 장점에 비해 단점이 전혀 없는 것은 아니었다.

먼저 프로그램을 로드할 때 〈F3〉을 누르면 나타나는 ＊.pas는 파스칼 이외의 언어에서는 오히려 불편하게 되어 있다. 가령 어셈블리어로 프로그램을 작성한 후 저장하기 위해 〈F2〉를 누르면 그 프로그램은 어셈블리어임에도 불구하고 확장자가 〈화일이름〉.pas로 저장된다. 다른 하나는 특정 컴파일러(터보 파스칼)에 의존적이라는 점이다. 물론 통합 환경용 에디터이기 때문에 당연한 일이겠지만 터보 파스칼 이외의 작업에서는 여간 성가신 것이 아니었다.

그외 아스키 그래픽 문자를 입력할 때 따르는 불편 (일일이 ALT키를 누르고 해당 아스키번호를 입력해야 한다) 등이 있다. 그래서 여러가지 목적으로 쓰일 수 있을 뿐만 아니라 기존의 터보 에디터로서의 장점을 모두 누릴 수 있도록 한 새로운 에디터를 만들게 되었다.

이 에디터는 터보 파스칼 5.0으로 작성하였다. 그러나 유니트 BBUNUTIL.PAS속의 OS_Shell 프로시듀어에 있는 펑션 GetEnv만 제외하고는 버전 4.0으로도 실행을 시킬 수 있다. 즉, 적당한 프로시듀어에 forward문을 이용한다면 4.0 컴파일러로도 컴파일을 시킬 수가 있다. 이 펑션의 기능은 환경 블럭의 내용을 읽어오는 것인데 4.0에서 이 기능을 이용하기 위해서는 다른 언어와 링크시켜 사용하거나 아니면 4.0내의 inline이라는 명령으로 어셈블리 루틴을 작성하면 된다. 간단한 방법은 GetEnv('COMSPEC') 대신에 'd:\COMMAND.COM'을 쓰면 된다.(여기서 d는 처음으로 부팅되는 드라이브명이다.)

처음에 BBUN 에디터를 만들었을 때는 모든 프로시듀어들을 메인 프로그램 속에 넣어 덩치 큰(?) 프로그램이 되고 말았다. 나중에 유니트(Unit) 구조의 편리함을 이용하기 위해 결국 이 구조로 바꾸어 BBUNUTIL, BBUNSCRN, BBUNMENU, BBUNBLOK, BBUNFILE, BBUNKEY의 6개 유니트로 분류하고 이들을 메인 프로그램 BBUN.PAS에서 uses절로 수집(Collection)하였다(리스트 1 참조).

```
program BBUN_Editor;
uses
   Crt,Dos,BBUNUTIL,BBUNSCRN, ···
procedure BlockProcess(var Point
  begin
   :
```

유니트는 상수, 데이터형, 변수, 프로시듀어, 함수들의 모임으로 각 유니트는 독립된 파스칼 프로그램과 거의 같은 성격을 유지하며 프로그램 작성의 효율은 물론 일반적인 에디터가 제공하는 것보다 더 큰 화일을 작성할 수 있게 해준다.

화면 1은 버전 5.0의 통합환경 속에서 BBUN의 실행화일(.EXE)을 얻기 위해 Destination을 디스크로 바꾸어 컴파일시키고 있는 것이다.

터보 파스칼은 다른 언어에 비해 독특한 매력을 가지고 있다. 원래 파스칼은 교육용 언어로 작성되었기 때문에 문법의 구조가 엄격하고 대신 이해하기는 편한 언어이다. 하지만 표준 파스칼은 교육용일뿐 실제적인 이용에는 부족한 점이 많았다. 이를 보완해서 터보 파스칼이 나오게 됐는데 그 기능은 실로 막강한 것이었다. 우리가 이용하는 그래픽 유틸리티 가운데 닥터할로우(DRHALO)라는 것이 있다. 이 유틸리티내의 거의 모든 명령어가 터보 파스칼 4.0이나 5.0에서 구현이 가능하다. 또 표준 파스칼에서는 생각할 수 없었던 시스템과의 밀착성(물론 C보다는 뒤떨어지지만)을 가지고 있다.

## 프로그램 구조

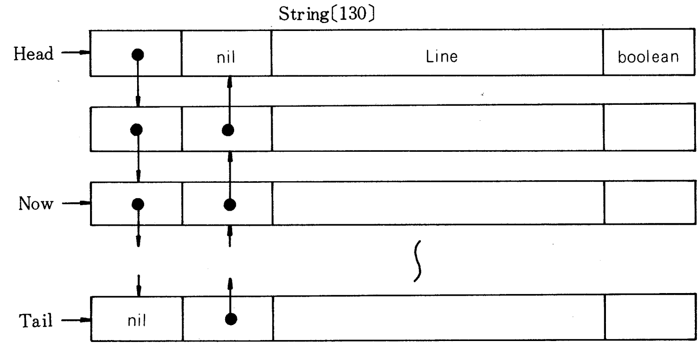
BBUN Editor는 5.0에서 제공하는 포인터 변수와 스트링을 이용하여 제작되었으며 그 구성은 그림 1에 보인 바와 같다. 즉 한 라인은 String[130]으로 최대 가능한 글자수를 130자로

〔화면 1〕 실행화일(.EXE)을 얻기 위한 컴파일 메뉴

```
  File    Edit    Run    Compile    Options    Debug    Break/watch
     Line 1     Col 8    | Compile  Alt-F9       | :EDIT5.PAS
{$M $4000, 0, 300000}    | Make         F9       |
program BBUN_Editor;     | Build                 |
                         | Destination   Disk    |
 uses                    | Find error            |
   Crt,Dos,BBUNUTIL,BBUNS| Primary file:         |KEY;
{------------------------| Get info              |----------------}
{-----------------------------------------------------------------}
 procedure BlockProcess(var Point_Var : PointType; var Cursor : CursorType;
   begin
     with T_F do
      begin
       if Point_Var.Head <> Point_Var.Tail then
        begin
         if not InsideBlock(Point_Var.Now,Point_Var.BlockStart,Point_Var.B
           and Point_Var.BlockStart^.Mark then
          if Move then
            MoveBlock(Point_Var,Cursor)
          else if Copy then
------------------------------- Watch -----------------------------

F1-Help  F5-Zoom  F6-Switch  F7-Trace  F8-Step  F9-Make  F10-Menu
```

〔그림 1〕 스트링 구성




놓았다. 만일 더 많은 수의 글자를 필요로 한다면 유니트 BBUNTIL 안의 MaxHorFile=130;을 적당한 값(255보다 작은)으로 바꾸어 주면 된다. 한 라인의 끝부분에 놓여 있는 boolean변수는 블럭 처리를 제어하는 변수로 이용된다.

그림 2는 BBUN 에디터의 구조를 간단히 그림으로 나타낸 것인데 여기서 큰 사각형은 화일을 나타내며 작은 사각형은 화면을 나타내고 있다. BBUN 에디터에서 화면은 화일 위를 움직이는 윈도우가 되는 것이다. 또한 커서의 위치는 화면 안에서 (X,Y)로 나타내며 실제 (RX+X-1, RY)로 표시된다. 여기서 RY는 별다른 기능을 수행하는 것이 아니라 화면 상단의 'Line : xx'를 나타내기 위한 것이다. 전체적인 프로그램의 흐름은 그림 3의 순서도를 참고하기 바란다.

라인과 라인은 포인터를 이용하여 연결시켰고 한 라인의 내용을 스트링에 담음으로써 프로그램은 쉽게 작성할 수 있었지만, 로드, 또는 저장할 때 속도가 떨어지는 점과 미리 화일의 크기와 메모리의 여분을 체크하고 비교할 수 없다는 점이 아쉽기는 하지만 간단한 에디터 기능으로는 별 문제가 없을 것으로 본다.

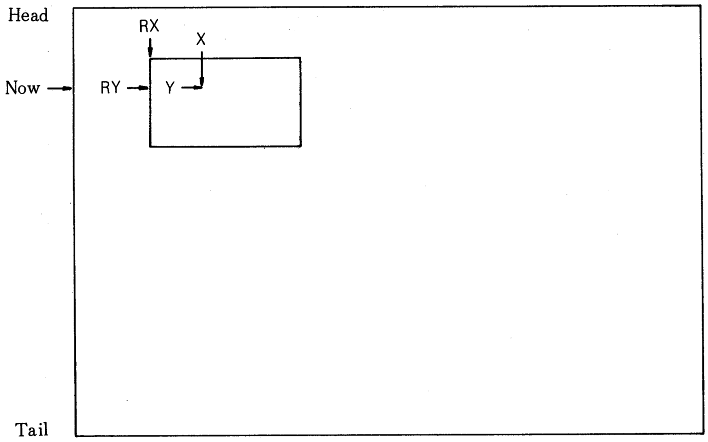
대신 5.0에서 제공되는 Insert, Delete, Concat, Dos 명령문을 이용하여 한 라인 안에서의 추가, 삭제가 편리하고 검색과 대치를 손쉽게 처리를 할 수 있게 하였다. 그리고 앞서 말했듯이 라인과 라인을 포인터 변수로 구현함으로써 블럭의 복사, 이동, 삭제 등을 유용하게 하였다.

위와 같은 에디터의 구조는 포인터를 이용한 선형이 아니라 우리의 눈에 보이는데로 라인별로 구분했으므로 원리를 이해하는데 큰 지장은 없을 것이다. 참고로 포인터 변수를 이용하는데 주의해야 할 점을 살펴보기로 하자. 터보 파스칼의 매뉴얼을 보면 Heap관리라는 것이 나와 있다. Heap라는 것은 터보 파스칼이 메모리를 관리하는 형태로서 포인터 변수가 바로 이 Heap내에 존재하고 있으므로 포인터 변수를 이용하기 위해서는 이 Heap의 구조를 먼저 알아야 한다.

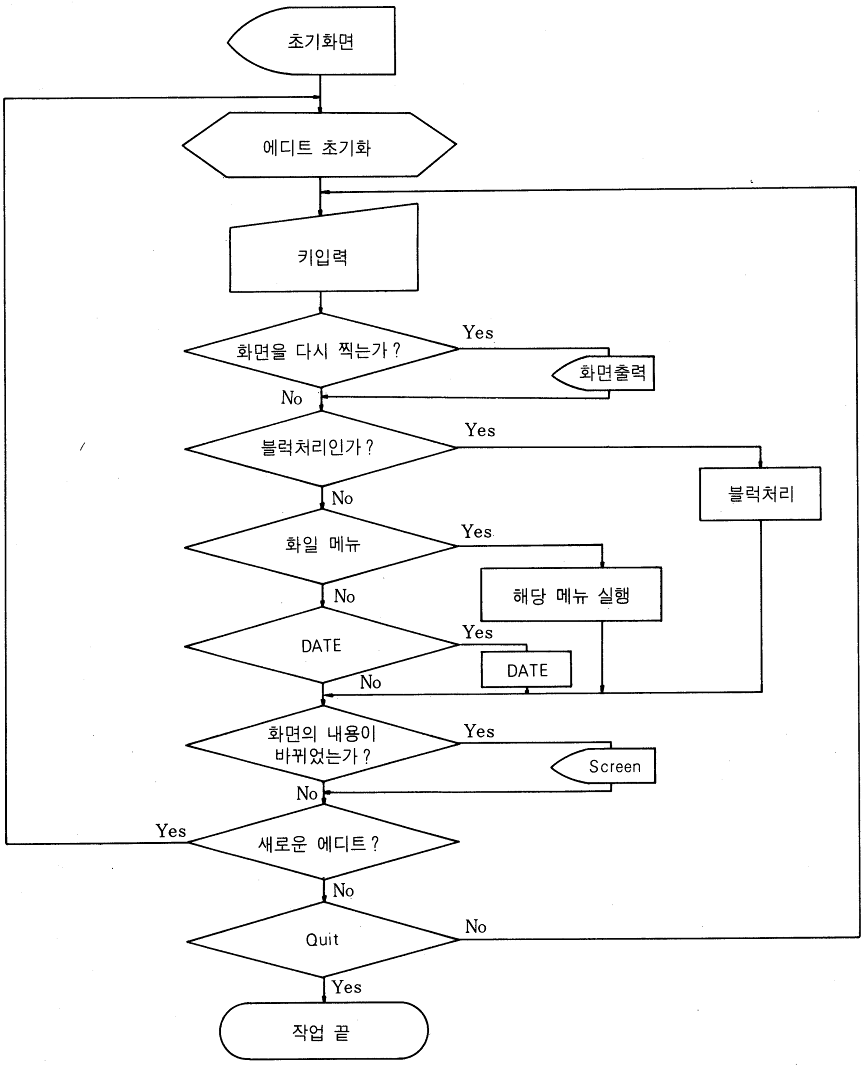
Heap내에서 포인터 변수를 제거할 때는 Dispose와 Release문을 이용하게 되는데 매뉴얼에는 이 두가지를 섞어서 사용하지 말라고 나와 있다. Heap의 끝 부분에는 freelist라는 부분이 있는데 바로 이 부분이 포인터 변수에 대한 정보를 담고 있다. 만일 release문을 사용한다면 이 부분까지도 클리어되어 버리기 때문에 BBUN 에디터에서는 포인터 변수를 지울 때는 Dispose문을 사용했고 프로그램을 초기화할 때는 Release(HeapOrg)를 사용했다. 여기서 HeapOrg는 Heap의 시작을 가리키는 포인터 변수로써 터보 파스칼에서 제공한다.

만일 Release (HeapOrg)를 사용한다면 Heap내의 모든 영역이 클리어된다. 프로그램에서 유니트 BBUNUTIL.PAS를 살펴보면 HeapFunc라는 펑션이 있는데 이것을 {$F+} 와 {$F-}를 사용하여 컴파일하고 HeapError:=@HeapFunc 로 선언해 두면 Get나 New로 포인터 변수를 생성할 때 Heap

〔그림 2〕 BBUN 에디터의 구조




〔그림 3〕 전체 순서도




이 부족하면 그 변수를 생성하는 대신에 그 변수에 아무 것도 없다는 의미의 nil값을 돌려준다. 이 값이 리턴되는 것은 HeapFunc내의 HeapFunc:=1; 이라는 것 때문이며 이 값은 변경될 수가 있는데 자세한 것은 터보 파스칼 매뉴얼을 참고하기 바란다.

## 기능

BBUN 에디터는 터보 파스칼 에디터를 기본으로 작성되었기 때문에 기능을 설명하기 위해서는 아무래도 터보 에디터와의 비교를 피할 수가 없을 것 같다.

### ●로드(Load)

터보 에디터의 경우에는 화일을 읽기 전에 메모리의 여분과 화일의 크기를 비교해서 화일의 크기가 더 크다면 'Truncate'를 표시한다. 하지만 BBUN 에디터는 한 라인이 스트링으로 되어 있기 때문에 화일을 읽는 도중에 메모리가 부족해지면 'Truncate'를 표시함과 동시에 자동으로 화일을 자른다. 한 라인이 스트링으로 되어 있기 때문에 한 글자도 최소한 130(MaxHorFile=130일때) 바이트를 차지한 상태가 된다. 그렇기 때문에 화일의 크기가 5000이라 하더라도 그 내용이 모두 리턴코드로 채워져 있다면 BBUN 에디터는 모두 읽지 못하게 된다.

### ●저장(Save)

화일을 저장하는 기능은 터보 에디터와 거의 같다. 〈F2〉로써 화일 이름을 묻지 않고 직접 디스크에 저장할 수 있으며 화일을 저장하는 도중에 디스크의 용량이 부족하면 'Disk is full'라는 에러 메시지와 함께 생성되던 화일을 지운다.

### ●블록의 로드와 저장

이 기능은 화일의 로드와 저장에 더

밀착되어 있기 때문에 블럭 처리와는 별도로 설명된다. 블럭을 읽어 들이고 싶을 경우에는 먼저 블럭을 표시한다. 그 다음 로드기능을 선택하면 로드 윈도우 위에 'Load BlockName'이라고 나올 것이다. 이때 읽어들일 블럭을 입력하면 표시된 블럭의 바로 밑에 이 블럭이 삽입된다. 그리고 블럭을 저장할 때는 블럭로드와 마찬가지로 블럭을 결정한 다음 저장기능을 수행하면 되는데 이때 직접적으로는 디스크에 저장되지 않는다는 것을 알아 두자. (BBUN 에디터에서는 터보 에디터에서처럼 블럭을 읽고 쓰는 기능을 담당하는 키가 따로 존재하지 않는다)

## ●정렬(Align)

이 기능은 터보 에디터에는 없는 기능으로 '정렬'이라고 했지만 사실은 간단하게 프로그램의 indent를 맞추어 주는 것이다. 보기 1과 같은 프로그램이 있을 때 마지막 줄을 C번 반복하고 싶을 때 while문을 이용한다고 하면 (터보 에디터의 경우는) 보기 2처럼 indent 가 맞추어지지 않는다. 이 경우는 while문장의 begin과 end를 포함해서 오른쪽으로 옮기기 위해서 각 라인마다 블랭크를 채워 넣는다면 (더구나 만일 블럭의 크기가 큰 경우는) 이 일은 상당한 인내를 필요한 하는 작업이 될 것이다. 이 기능은 이런 수고를 덜어주는데 먼저 블럭을 정한 다음, 이 기능을 택하면 공백이 삽입될 컬럼과 갯수를 묻게 되는데 적당한 수를 입력하면 그 블럭은 indent가 자동적으로 맞추어지게 된다. 또한 공백이 삽입될 갯수는 음수도 가능하다.(음수의 경우는 삽입이 아니라 삭제가 된다.)

## ●찾기, 대치(Find, Replace)

터보 에디터를 사용해 본 독자라면 알겠지만 터보 에디터는 Find기능의 처리에 있어서 매우 불만족스러운 부분이 있다. 또한 Replace에 비해서 Find

〔보기 1〕

```
A:=1;
B:=2;
C:=3;
Inc(A, B);
```

〔보기 2〕

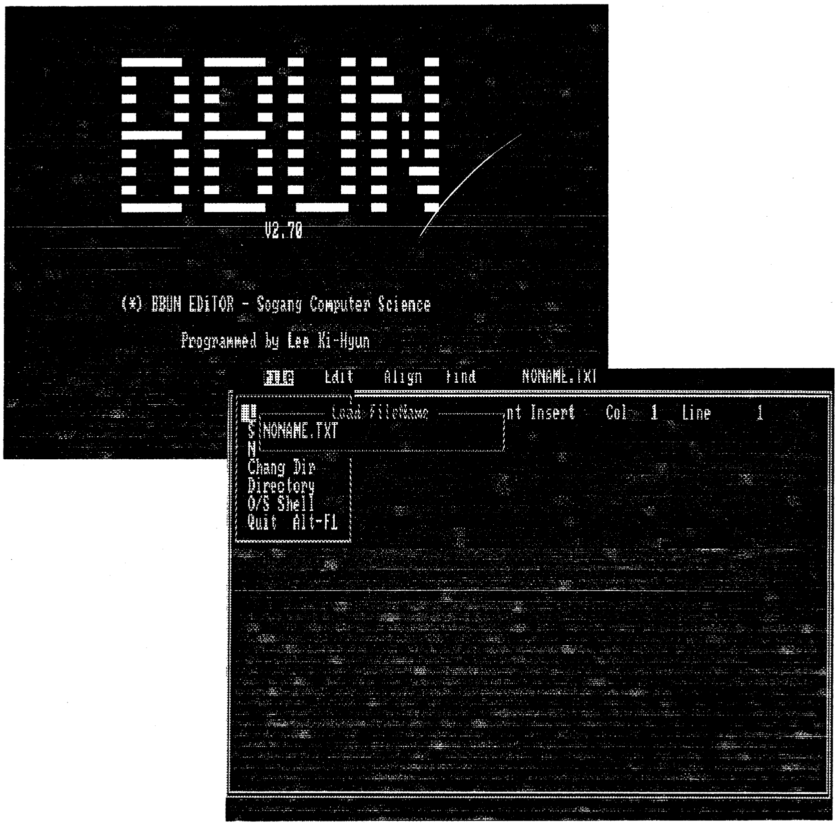
```
A:=1;
B:=2;
C:=3;
while (C>0) do
begin
Inc(A, B);
Dec(C)
end;
```

기능을 성실하게 수행하지도 않으며 이 두가지 기능을 수행할 때는 Option 이라는 불필요한 코맨드가 항상 붙어 다닌다. BBUN 에디터는 이 두가지 기능을 수행할 때 Find 또는 Replace할 단어를 물어온다. 이를 입력하고 나서 찾아 나가는데 이때 한 단어를 찾을 때마다 윈도우의 상단에 'Find:xx'라고 해서 찾은 수를 표시해 준다. 이 수를 보면서 자기가 적당한 수를 찾았다고 생각되면 〈ESC〉로써 기능을 중지시키면 된다.

## ●ASCII 문자 입력

이 기능은 터보 에디터에서 보다 향상된 것으로 터보 에디터의 경우 그래픽 문자를 그리기 위해서는 〈ALT〉키와 동시에 그 문자의 아스키 값을 입력해야 했다. 한두 문자를 입력할 경우에는 그럭 저럭 사용할 수 있겠지만 여러개의 그래픽 문자를 필요로 할 경우에는 지루한 작업이다. BBUN 에디터에서는 〈F6〉키를 누르면 윈도우상단에 그래픽문자들이 나타나면서 커서

〔BBUN 에디터의 초기화면과 메뉴선택〕



가 생기고 원래 커서가 있던 자리는 반전됨과 동시에 반짝거리고 있게 된다. 이때 적당한 그래픽문자를 택하고 리턴을 치면 해당하는 자리에 그래픽문자가 찍히게 된다. (찍히게 되는 그래픽문자는 그때의 화면모드(Insert)에 따르게 된다) 필자는 프로그램에 직접 그래픽문자를 집어 넣는 것을 꺼리는 편인데 필자와 같은 독자가 있다면 리턴키 대신에 〈ESC〉를 치면 해당 그래픽문자와 함께 아스키값이 10진수와 16진수로 표시될 것이다. 물론 화면에는 찍히지 않는다.

터보 에디터에서 만일 그래픽문자로 사각형을 그린다면 여러개의 '─'를 찍기 위해 〈ALT〉-1-9-6을 여러번 쳐야 할 것이다. '─'을 5번 찍기 위해서는 실제적으로 키를 15번 쳐야 한다. BBUN 에디터에서는 '─'을 한번 찍은 후에 〈CTRL〉-A를 누르고 있으면 라인이 그려질 것이다. 〈CTRL〉-A는 아스키코드의 입력과 함께 사용하기 위하여 만든 것이지만 만일 같은 글자를 여러번 찍고 싶다면 그 문자를 화면에 찍은 후 〈CTRL〉-A를 이용하면 된다. 이 키는 화면에 찍은 문자만을 반복해서 찍어주므로 〈ALT〉-K를 누른 후 이 키를 이용하면 그전에 화면에 찍혔던 문자가 찍히게 된다.

## ●날짜(Date)

프로그램 작성 도중에 〈ALT〉-O를 누르면 현재 시스템에 들어 있는 날짜를 표시해 준다.

## ●라인 복구(Undo)

터보 에디터의 경우에는 전에 삭제된 라인을 복구할 때 라인을 삽입한 후에 undo를 한다. BBUN 에디터에서는 이 경우 〈ALT〉-E만 눌러주면 자동으로 한 라인이 삽입되면서 undo기능이 수행된다. 그리고 복구되는 라인은 가장 최근에 지워진 라인을 원칙으로 한다. undo기능이 반드시 지워진 라인만을 복구하는 것이 아니라 한

〔리스트 1〕 메인프로그램(BBUN.PAS)

```
{$M $4000, 0, 300000} (* 스택과 힙의 크기를 정한다 *)
program BBUN_Editor;

 uses
   Crt,Dos,BBUNUTIL,BBUNSCRN,BBUNMENU,BBUNBLOK,BBUNFILE,BBUNKEY;

  procedure BlockProcess(var Point_Var : PointType; var Cursor : CursorType; var T_F :
   begin                                                                     TFType);
     with T_F do
      begin
        with Point_Var do
         begin
           if Head<>Tail then
            begin
              if BlockStart^.Mark and                            (* 블럭이 정해져있고 *)
                 not InsideBlock(Now,BlockStart,BlockEnd) then   (* 커서가 블럭안에 없다면 *)
                if Move then
                  MoveBlock(Point_Var,Cursor)                    (* 블럭이동 *)
                else if Copy then
                  CopyBlock(Point_Var)                           (* 블럭복사 *)
                else if Delete then
                  DelBlock(Point_Var,Cursor);                    (* 블럭삭제 *)
              if DelVar then
                VarDel(Point_Var,Cursor)                         (* 라인삭제 *)
            end;
           if InsVar then
             VarIns(Point_Var,Cursor)                            (* 라인삽입 *)
        end;   (* with Point_Var *)
        if Screen then       (* 블럭처리가 완전하면 Screen은 True값을 갖는다 *)
          Undo.Line := Point_Var.Now^.Data;
        Move := False;   Copy := False;
        Delete := False;  InsVar := False;
        DelVar := False;
        T_F.Update := True
      end     (* with T_F *)
   end; (* end of BlockProcess *)


    function FileProcess(var Point_Var : PointType;
                         var T_F      : TFType;
                         var FileName  : string) : boolean;
    var
      Check      : boolean;
    begin
      Check := False;
      with T_F do
       begin
         if Directory then
           Directory_(T_F,Check)                  (* 디렉토리 보기 *)
         else if ChDir_ then
           Chg_Dir(T_F,Check)                     (* 디렉토리 변경 *)
```

라인의 내용이 매우 복잡할때는 이 라인을 수정하는 도중에 다시 이 라인을 복구할 필요가 생길 때가 있다. 이때는 〈ALT〉-U를 눌러주면 커서가 이 라인에 처음 위치했을 때의 라인상태로 환원된다.

### ●블럭 처리(Block Process)

BBUN 에디터에서 블럭의 처리는 터보 에디터와 거의 비슷하지만 약간 틀린점이 있다. 터보 에디터의 경우에는 블럭의 이동과 삭제시에 한줄의 공백라인이 생긴다. 하지만 BBUN 에디터에서는 이 경우 공백라인이 생기지 않으며 이동과 복사시에 한 줄의 공백라인을 삽입할 필요없이 커서가 위치한 라인의 바로 위에 삽입시킬 수가 있다.

### ●에러 처리(Error)

BBUN 에디터는 기존의, 에디터(상업적인 에디터는 제외하고 여러 잡지에 실린 것)보다는 에러처리에 있어서 친절한 면을 제공한다. 프로그램내에서 IOResult와 DosError를 이용해서 에러처리를 하고 각 프로시듀어마다 요구하는 에러 메시지가 다르기 때문에 에러를 표시하는 부분이 조금 복잡하다. 나타나는 에러 메시지는 제한되었으나 더욱 확장하고 싶은 독자는 프로시듀어 ErrorMsg(ErrorCode : byte-); 내의 내용을 변경하거나 추가하면 될 것이다.

## 유니트 설명

### ●UNIT BBUNMENU

먼저 OS Shell을 담당하는 OS Shell procedure가 있는데 여기서 GetEnv라는 함수가 있다. COMSPEC이라는 환경변수의 Shell program을 읽어들이는 것으로 COMMAND.COM이 읽혀진다. 그리고 Align에서 찾거나 대체할 때 최대 갯수로 65000을 주었다. 이것이 부족하다고 생각되면 프로시듀어에서

```
        else if Load then
          FileLoad(Point_Var,FileName,Check)    (* 로드 *)
        else if Save then
          FileSave(Point_Var,FileName,Check);   (* 세이브 *)
        FileProcess := Check (* 로드, 세이브가 완전하면 Check는 True 값을 갖는다 *)
      end; (* with T_F *)
    end; (* end of FileProcess *)


  procedure ShiftUp(var T_F : TFType);
  var
   Trans : boolean;
begin
  Trans := T_F.Load;              T_F.Load := T_F.Save;
  T_F.Save := T_F.New;            T_F.New := T_F.ChDir_;
  T_F.ChDir_ := T_F.Directory;    T_F.Directory := T_F.Shell;
  T_F.Shell := T_F.Quit;          T_F.Quit := Trans
end;

procedure ShiftDown(var T_F : TFType);
var
   Trans : boolean;
begin
  Trans := T_F.Quit;              T_F.Quit := T_F.Shell;
  T_F.Shell := T_F.Directory;     T_F.Directory := T_F.ChDir_;
  T_F.ChDir_ := T_F.New;          T_F.New := T_F.Save;
  T_F.Save := T_F.Load;           T_F.Load := Trans
end;

procedure ShiftLeft(var T_F : TFType);
var
   Trans : boolean;
begin
  Trans := T_F.FileMenu;      T_F.FileMenu := T_F.Edit;
  T_F.Edit := T_F.Align;      T_F.Align := T_F.Find;
  T_F.Find := Trans
end;

procedure ShiftRight(var T_F : TFType);
var
   Trans : boolean;
begin
  Trans := T_F.Find;       T_F.Find := T_F.Align;
  T_F.Align := T_F.Edit;   T_F.Edit := T_F.FileMenu;
  T_F.FileMenu := Trans
end;


procedure Popup(var T_F : TFType);
var
   Count : byte;
   Ch    : char;
```

Conunt의 Type과 마지막 부분 unit문에서 (count=65000)을 적당한 값으로 변경시켜 주면 된다.

### ●UNIT BBUNBLOK

블럭처리에 관한 부분으로 이동 및 삭제, 복사와 라인의 삭제, 삽입이 행해진다.

### ●UNIT BBUNSCRN

화면처리에 관한 부분으로 상단의 메뉴 및 팝업메뉴, 그리고 화면의 내용을 출력해 준다.

### ●UNIT BBUNFILE

ChDir: 현재의 디렉토리를 바꾸어 준다.

Directory: 디렉토리 내의 화일 및 서브 디렉토리를 소트하여 표시한다.

File Load: 화일을 로드하는 부분으로 각종 에러처리와 truncate를 담당한다.

File Save: 저장에는 직접방식과 간접방식이 있는데 전자의 경우는 〈F2〉키로써 화일이름을 묻지 않고 직접 저장하며 후자의 경우는 메뉴를 통한 것으로 화일의 이름을 물어 온다.

BBun key: 에디터내에서 실질적인 부분으로 각종 키의 입력과 동시에 프로시듀어를 부르기 위한 boolean변수의 세트와 동시에 리턴 키, 백스페이스 키, 탭 키 등을 처리해 준다.

BBun: 메인 부분으로 여러 프로시듀어를 부르는 부분이다. 팝업 메뉴처리 부분으로 Quit나 New가 실행될 때까지 루프를 수행한다.

## 프로시듀어 설명

### ●Initvalue( )

모든 변수를 세트시켜 주며 boolean 변수에서 Insert와 Indent는 디폴트로 켜져 있다.

### ●Inverse, Normal

화면에 문자를 찍을 때 반전 또는

```
begin
  with T_F do
   begin
     if not Direct then
       Load := True;
     repeat
       DrawMenu(T_F,FileName);
       if not Direct then
         Ch := ReadKey
       else
         Ch := _Return;
       if (Ch=_Nul) or (Ch=_Return) then
        begin
          if (Ch=_Nul) then
            Ch := ReadKey;
          if Ch=_Down then
            Ch := _Return;
          Case Ch of
            _Left   : ShiftLeft(T_F);
           _Right   : ShiftRight(T_F);
          _Return   : if FileMenu then
               begin
                 PushWindow(3,3,16,9,'',Window_);
                 repeat
                   repeat
                     DrawPopUp(T_F);
                     if not Direct then
                       Ch := ReadKey;
                     if Ch=_Esc then
                       Ch := _Nul
                     else if Ch=_Nul then
                      begin
                        Ch := ReadKey;
                        Case Ch of
                          _Up   :  ShiftUp(T_F);
                          _Down :  ShiftDown(T_F);
                          _Left : begin
                                    ShiftLeft(T_F);
                                    Ch := _Nul
                                  end;
                         _Right : begin
                                    ShiftRight(T_F);
                                    Ch := _Nul
                                  end;
                          _F3   : begin
                                    while not Load do
                                      ShiftUp(T_F);
                                    Ch := _Return
                                  end;
                          _F2   : begin
                                    while not Save do
                                      ShiftUp(T_F);
```

정상을 나타낸다.

## ●Cursor_on, Cursor_off

화면에 커서를 나타내거나 사라지게 한다.

## ●Click

산뜻한 클릭음을 낸다.

## ●DisableInterrupts

터보 파스칼에서 Check Break라는 변수가 있다. 따라서 처음 세트시에 Check Break:=False라고 하면 그 후로는 CTRL-Break가 걸리지 않는다. 그러나 DOS를 처음 부팅할 때 break=on이라고 선언을 해주면 CTRL-Break는 걸리지 않지만 CTRL-C는 인터럽트를 걸게 한다. 만일 로드나 저장할 때 CTRL-C를 누른다면 최악의 경우 화일이 깨지는 경우가 생긴다. 필자 자신도 CTRL-C 때문에 이 에디터를 사용하다가 프로그램을 깨먹은 적이 있는데, 마침 DOS에서 CTRL-C를 막아주는 인터럽트가 있어서 이 인터럽트로 CTRL-C를 막을 수가 있었다.

## ●function Wordlength

파스칼에서 length는 스트링의 길이를 나타내 준다. 하지만 마지막의 공백도 포함시키게 되므로 'ABC '의 길이는 4가 된다. 이를 없애려고 실질적인 유효자리만을 세는 것이 wordlength이다.

## ●function Upcase_

터보 에디터에서 Upcase는 char형만을 받아들인다. 이를 변형한 것으로 string type의 모든 변수를 대문자로 변환하여 준다.

## ●function Deadkey_

2자리 값을 가지는 스캔 키의 값을 Null로 바꾸어 주고 그 이외의 키는 아스키값만 읽어 들인다.

(예)F1 등의 함수키와 ALT 등의

```
                              Ch := _Return
                            end
                        else     Ch := ' '
                      end
                    end;
                 until (Ch=_Return) or (Ch=_Nul);
                 if Ch=_Return then
                   if Shell then
                     OS_Shell(T_F)
                   else if Directory or ChDir_ or Save or Load then
                      if not FileProcess(Point_Var,T_F,FileName) then
                        Ch := _Esc;
                 Direct := False;
               until (Ch=_Return) or (Ch=_Nul);
               PopWindow(Window_)
             end;
     _F3,_F2 : begin
                repeat
                  ShiftLeft(T_F)
                until FileMenu;
                DrawMenu(T_F,FileName);
                if Ch=_F3 then
                  Load := True
                else
                  Save := True
              end
    end  (* end of case *)
  end
until (Ch=_Return) or (Ch=_Down) or (Ch=_Esc);
  if Ch=_Return then
    if Find then
          Find_Replace(Point_Var,Cursor,T_F,F_R)
        else if (Align) and (Point_Var.BlockStart^.Mark) then
          Align_(Point_Var,T_F);
    if not ((New or Quit) and (Ch=_Return)) then
     begin
       Count := 1;
       while (not Load) and (Count<8) do
        begin
          ShiftUp(T_F);
          Inc(Count)
        end;
       Load := False
     end;
    Count := 1;
    while (not FileMenu) and (Count<5) do
     begin
       ShiftLeft(T_F);
       Inc(Count)
     end;
    FileMenu := False;
    Menu := FileMenu;
```

키는 두자리 값(0;?)의 스캔 코드를 갖는데 이것을 0으로 바꾼다.

**●ADDBLANK**

공백을 더하여 준다.

**●fillblank**

현재의 커서 위치를 기준으로 공백을 채워 넣는다.

**●Removeblank**

length와 wordlength의 결과가 같도록 마지막 부분의 공백을 삭제한다.

**●function Inside block(A, B, C)**

포인터 변수를 블럭 처리시 사용할 때 이용하는 것으로 만일 세 포인터 변수 A, B, C가 B<A<C로 (B와 C사이에 A가 위치해 있다) 되어 있으면 True값을 돌린다.

**●procedure offset**

화면에 직접 액세스할 때 현재의 커서 위치를 기준으로 오프셋 번지를 얻는다.

**●Pushwindow, popwindow**

윈도우를 만들거나 해제한다. 모든 윈도우가 BOX를 필요하므로 pushwindow 는 BOX를 그린다.

**●Readline**

현재의 커서 위치로부터 스트링을 읽어들이기 위한 것으로 터보 에디터에서의 팝업메뉴에서 키의 입력과 유사하게 작용한다.

**●Date**

년.월.일.요일을 나타내며 수정을 할 수 없게 되어 있다.

## 키 설명

F1 : HELP 기능을 가지고 있으며 간결하게 키와 기능을 디스플레이 한다.

```
      DrawMenu(T_F,FileName)
    end (* with T_F *)
  end; (* end of Popup *)



begin
  DisableInterrupts;   (* Ctrl-C 인터럽트를 막는다 *)
  CheckBreak := False; (* Ctrl-Break 인터럽트를 막는다 *)
  Window_.X1[1] := 0;        Window_.X1[2] := 0;
  Window_.X1[3] := 0;        Window_.X1[4] := 0;
  Window_.Y1[1] := 0;   (* 팝업메뉴를 위한 윈도우를 초기화 시킨다 *)
  CursorOff;
  Normal;
  with T_F do
   begin
     New := True;
     while New do
      begin
        InitValue(Point_Var,Cursor,T_F,F_R,FileName);
        LastKey := ' ';   (* Ctrl-A *)
        ClrScr;
        DrawMenu(T_F,FileName);
        DrawBox;
        Display(Cursor,T_F);
        if Window_.Y1[1]=0 then
          InitScreen;      (* InitScreen은 단 한번만 불리우게된다 *)
        Click;
        while not(Quit or New) do
         begin
           KeyRead(Point_Var,Cursor,T_F,LastKey); (* 키입력 *)
           if Help then
             HelpKey;
           Display(Cursor,T_F);
           if Line or Screen then
             PrintScreen(Point_Var,Cursor,T_F);
           if Move or Copy or Delete or InsVar or DelVar then
             BlockProcess(Point_Var,Cursor,T_F)
           else if Menu or Load or Save or Align or Find then
             Popup(T_F)
           else if Date then
             Date_(T_F);
           if Line or Screen then
             PrintScreen(Point_Var,Cursor,T_F)
         end;
        if Update then            (* 저장되지 않은 화일처리 *)
         begin
           ErrorMsg(200);
           repeat
             LastKey := Upcase(ReadKey_)
           until (LastKey='N') or (LastKey='Y');
           if LastKey='Y' then
```

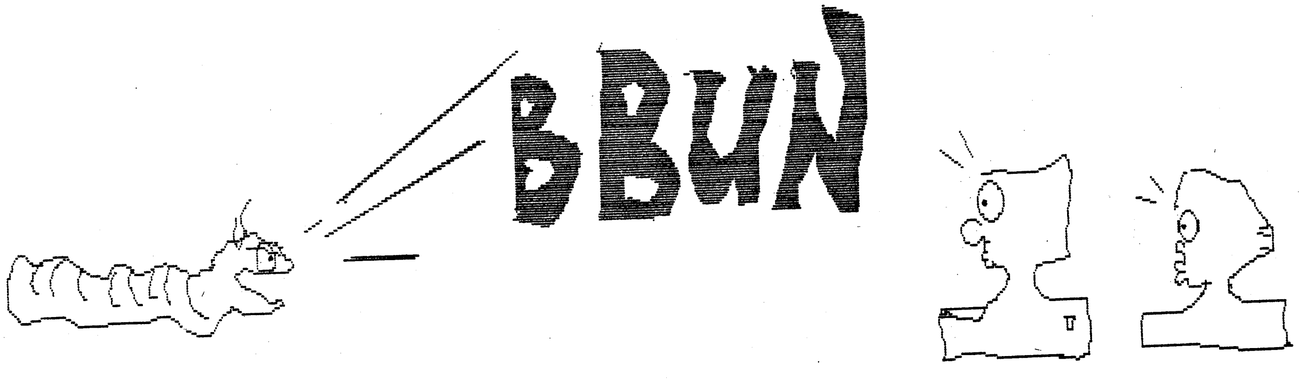
**F2**: 터보 에디터와 동일하게 세이브하는 기능을 가지고 있으며 팝업메뉴를 거치지 않고 직접 세이브 기능을 수행할 수도 있다. 블럭이 표시되어 있으면 블럭 세이브를 수행한다.
**F3**: 로드기능을 수행하며 블럭이 표시되어 있으면 블럭 로드가 수행된다.
**F4**: New기능을 수행한다.
**F5**: Align기능으로 메뉴를 거치지 않는다.
**F6**: 아스키 코드 입력으로 메뉴를 거치지 않는다.
**F7**: FIND와 REPLACE를 수행하며 메뉴를 거칠 수도 있다.
**F8, F9**: 각각 블럭의 처음과 끝부분을 표시하며 이때 해당 블럭은 반전된다.
**F10**: 메뉴를 부르며 화면 상단의 메뉴 중 'File'이 반전된다.
**ALT-F1**: Quit
**CTRL-Home, End**: 화일의 처음과 끝으로 이용한다.
**ALT-I**: Indent기능을 토글시킨다.
**ALT-E**: undo를 수행하며 바로 전에 지워진 라인을 복구한다.
**ALT-U**: undo기능으로 지금 작업중인 라인을 복구한다.
**ALT-H**: 표시된 블럭을 반전시키거나 정상 상태로 복구한다.
**ALT-M**: 블럭을 표시한다.
**ALT-C**: 블럭을 복사한다.

```
            begin
              Save := True;
              Load := FileProcess(Point_Var,T_F,FileName)
            end;
           PopWindow(Window_)
         end;
        Release(HeapOrg)         (* 힙을 클리어한다 *)
      end
    end;
   Window(1,1,80,25);
   EnableInterrupts;
   CheckBreak := True;
   CursorOn;
   ClrScr
end.
```

**ALT-D**: 블럭을 삭제한다. 만일 블럭이 표시되지 않았으면 ALT-Y와 같은 기능인 라인 삭제를 수행한다.
**ALT-N**: 라인을 삽입함

BBUN 에디터는 팝업메뉴와 메뉴의 테두리를 그릴 때 메모리를 직접 액세스하므로 허큘리스카드만을 지원한다. 만일 허큘리스가 없다면 유니트 BBUNUTIL.PAS의 Segmen=$b000을 자기가 소유하고 있는 비디오 카드에서 텍스트 모드의 시작번지로 바꾸어 주면 된다. 또 BBUN 에디터는 프로시듀어를 부를 때 boolean 변수를 이용하므로 만일 독자가 필요한 기능이 있다면 그에 해당하는 boolean변수와 프로시듀어를 추가하면 된다. 아무래도 필자의 프로그래밍이 서툰 관계로 프로그램이 어수선하고 부족한 점이 많을 것이다. 좋은 프로그램은 그 자체가 좋은 설명이라고 했는데 과히 좋은 프로그램이 아니기 때문에 군데 군데 주석문을 달아 놓았다.

필자의 이름 옆에 붙어있는 'BBUNDEX' 는 필자가 속한 스터디(전산과내) 이름이고 별다른 뜻을 가지고 있는 것이 아니라 우리말로 소리나는대로 영어로 옮긴 것이다.(약간 우수꽝스럽기는 하지만) 마지막으로 필자가 이 프로그램을 만드는데 도와준 BBUNDEX에게 감사드린다.




(알림) BBUN 에디터의 나머지 6개 유니트화일은 지면관계상 모두 싣지 못하고 디스켓으로 제공할 예정입니다.

# 음성의 저장과 재생이 가능한
# 음성응답장치

**김원준**/건국대학교 전자공학과 3년

근래에 들어 음성처리에 관한 관심도가 급속히 높아짐에 따라 우리의 일상 생활속에서도 쉽게 음성처리기술을 응용한 각종 음성서비스들을 찾아 볼 수 있게 되었다. 지금까지 주를 이루고 있던 문자나 기호에 의한 정보에 비해 음성정보가 갖는 장점은 우선 인간과의 친밀도가 높다는 점을 들 수 있다. 또한 음성을 이용한 정보는 특별한 교육을 받지 않은 사람들도 쉽게 이해할 수 있다. 이러한 이유로 인하여 음성정보와 음성서비스의 사용율은 급속하게 증가되고 있다.

예를 들어 이미 오래전부터 시행되고 있는 전화 116번의 시간 음성안내, 그리고 요즈음 거리에 등장한 '파란불이 켜졌읍니다.'하는 신호 등 안내, 또한 상점 등에 설치된 '어서 오십시오'하는 안내장치, 무인 응답전화기 등이 그것이다. 누구나 한번쯤 이러한 장치에 대한 호기심을 가져 보았을 것이다.

특히 음성메시지를 출력해주는 장치를 직접 제작하여 자신의 음성이 흘러나오는 것을 들었을 때의 느낌은 오디오용 녹음기에서 느끼는 감동과는 조금 색다른 매력을 안겨 줄 것이다. 또 한가지 부언하고 싶은 것은 시각장애자에게 음성정보는 더할 수 없는 도움이 될 것이라는 점이다.

하드웨어를 공부하는 분들께는 한번쯤 음성저장과 재생에 관하여 공부해 볼 것을 권하고 싶다. 왜냐하면 음성의 저장과 재생은 그리 고도의 기술을 필요로 하지 않으면서도 다양한 기술들을 필요로 하기 때문에 실제로 도전해 보면 그만큼 많은 분야에 대한 공부를 할 수 있기 때문이다.

즉, 음성을 디지탈적인 방법으로 저장하고 재생하기 위해서는 우선 아날로그/디지탈 변환기술이 필요하고, 신호원이 음성이므로 필요없는 주파수 대역을 제한하기 위한 필터를 구성할 줄 알아야 하며, 메모리관리의 효율성을 높여야 한다. 그 밖에도 이를 처리하기 위한 하드웨어와 소프트웨어에 관한 전반에 걸친 지식이 필요하다. 한마디로 음성의 저장과 재생은 디지탈 녹음기를 만드는 것이다.

디지탈 녹음기는 기계적인 동작부가 필요치 않기 때문에 당연히 수명이 길고, 사용에 따라 일어날 수 있는 저장된 데이터의 음질손상도 없게 된다. 또한 소형화할 수 있으며, 다양한 기능을 갖출 수 있다. 그리고 랜덤 액세스가 가능하다. 일반 테이프처럼 앞의 곡을 듣기 위해 다시 테이프를 돌리거나 할 필요없이 원하는 메시지를 고속으로 출력하는 것도 가능하다. 뿐만 아니라 음성의 저장, 재생, 속도조절 등의 많은 장점을 갖고 있다.

반면에 메모리용량이 많이 소모된다는 단점도 갖고 있다. 그 이유는 뒤에서 이야기하겠지만 음성신호를 저장하기 위해서는 약 7kHz의 샘플링이 필요한데, 이것은 1초당 7k바이트로 상당히 많은 양의 메모리용량을 소모한다(8비트 저장의 경우). 게다가 일반 A/D 콘버터를 사용할 경우 고속 샘플링이 불가능하여(10kHz 정도가 한계) 고음에 둔해진다.

이러한 단점에도 불구하고 아날로그 녹음기에 비해 사용상의 편리성과 고신뢰도로 인해 간단한 음성메시지 출력용으로 상당히 많이 사용되고 있으며 앞으로의 전망도 매우 밝은 편이다. 이러한 음성신호의 저장, 재생기술은 음성인식의 기초지식이다.

여기서 소개할 음성응답장치의 기본원리는 본지 89년 1월호에 게재되었던 '음성출력시계'와 같은 맥락이기 때문에 이를 관심있게 읽은 독자들은 쉽게 이해할 수 있을 것이다. 따라서 중복되는 내용은 피하도록 하겠다. (뒤늦게 1월호의 '음성출력시계'를 읽게 되어 본 내용의 상당부분을 삭제하게 되었음을 양해하기 바란다.) 자세한 원리와 A/D변환에 관한 사항, 그리고 IBM이 아닌 타기종에서 개발하기를 원하는 독자는 '월간 컴퓨터학습' 88년 8월호에 소개된 '음성합성 장치의 제작'을 참고하면 좋을 것이다. 그리고 여기서 소개하는 회로는 교육용으로 제작되어 사용되고 있는 장치로 가능한한 쉽게 표현하겠지만 관련된 모든 내용을 언급하기는 어려우며 개발툴에 관한 이야기는 간단히 소개하는 정도로 그치겠다. 그러나 개발툴에 관한 정보는 시중의 여러서적에서 구할 수 있다.

## 구상동기

처음에 음성응답장치를 만들게 된 것은 학과 전시회에 사용하기 위해 '전자앵무새'를 만든게 계기가 되었다. 이 전자앵무새는 전시회를 구경하던

사람들이 앵무새에게 무어라고 이야기하면 그대로 따라서 말하는 장치로, A/D 콘버터로 음성을 입력받아 메모리에 저장한 후 다시 메모리에 저장된 데이터를 D/A 콘버터로 재생시켜 앵무새가 똑같은 음성을 출력하게 만든 것이다.

이것은 녹음기와 같은 원리를 이용하였다. 그러나 앞서 이야기한 바와 같이 유동적인 음성데이터를 저장하고 즉시 재생하는데 기계식 녹음기를 사용한다는 것은 무리이며, 또 그러한 녹음기를 인간의 조작없이 제어하기는 어렵기 때문에 디지탈방식을 사용하는 편이 낫다.

디지탈방식을 채택할 경우 가장 중요한 기술은 A/D, D/A 변환기술이다. 그러나 실제 어려웠던 부분은 필터였다. 이 부분은 회로에서 발생되는 고주파성분과 노이즈를 제거하는 회로로 그리 큰 효과를 느끼지도 못하고 사람을 찜찜(?)하게 한 고약한 것이었다.

이러한 '전자 앵무새'나 '음성출력시계' 등은 램을 이용하여 자유롭게 저장, 재생할 수 있으나 기억된 데이터를 유지하려면 항상 전원이 공급되어야만 한다. 그래서 실제 생활에서 활용할 수 있도록 하자면 데이터를 유지시켜 줄 필요가 있다. 데이터를 유지시키는데에 배터리 백업을 생각할 수있으나 이 방법은 항상 위험이 따른다.

따라서 가장 간단하고 확실한 방법은 롬을 사용하는 것이기 때문에 여기서 소개할 음성응답장치는 외부 롬을 이용하여 음성메시지를 저장해 두었다가 여러 음성안내를 제공할 수 있는 장치로, 컴퓨터와의 인터페이스가 아닌 독립된 시스템으로 제작한다. 앞서 말한 바와 같이 실용성을 위주로 하였기 때문이다.

음성메시지의 선택은 자체 키패드를 사용하거나 입력포트를 통하여 외부장치로 부터 선택되도록 한다. 그리고 LED 출력을 확인할 수 있도록 하였다. 또한 음성처리의 가장 큰 문제인 메모리를 쉽게 확장할 수 있도록 하였다. (제작한 회로는 최대 256K바이트의 데이터롬을 가질 수 있게 되어 있다.) 편의상 음성신호의 저장과 편집은 IBM과 인터페이스를 통해 하도록 하였다.

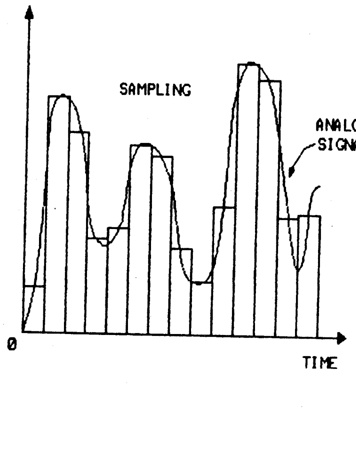
## 음성신호의 저장원리

음성신호는 약 3.5kHz 이내에서 변화하는 주파수와 진폭을 가진 신호이다. 이러한 음성신호를 컴퓨터에 저장하기 위해서는 아날로그 음성신호를 디지탈신호로 바꾸어 주어야 한다.

즉 그림 1의 곡선과 같은 파형을 갖는 음성신호가 입력된 경우 A/D 콘버터로 일정한 시간주기로 그때 그때의 신호의 진폭을 기록한다(샘플링). 이 진폭의 크기 또한 디지탈로 저장하게 되는데, 이 진폭을 얼마나 많이 나눌 수 있는가 하는 것이 A/D 콘버터의 해상도이다. 만일 8비트의 해상도를 가진 A/D 콘버터라면 최저 진폭인 경우 0, 최대 진폭인 경우 255가 될 것이다. 그림 1에서 볼 수 있는 막대그래프는 샘플링한 값을 나타낸다.

사람의 가청주파수 한계는 약 20kHz 라고 하지만 우리의 음성은 거의 3.5kHz 이내에서 변화한다. 그리고 공학적으로 신호를 큰 변화없이 샘플링 하였다가 재생하기 위해서는 그 신호가 가지는 최대 주파수 2배이상의 속도로 샘플링하여야 한다고 한다. 그러므로

〔그림 1〕 음성신호의 샘플링




음성을 샘플링하기 위해서는 약 7kHz의 속도가 필요하다.

반대로 재생하기 위해서는 D/A 콘버터가 필요하다. 이것은 기록된 디지탈신호를 다시 아날로그신호로 바꾸어 주는데, 처음의 아날로그신호와는 약간의 차이를 보이지만, 그림 1에서 막대의 폭이 좁을수록(샘플링 주기가 짧을수록) 정확한 재생이 가능하다. 이와 같이 음성신호의 저장과 재생은 일반적인 A/D, D/A 변환의 원리와 같은 것이지만 처리대상 신호가 음성이라는 점이 특이할 뿐이다.

## 제작과 회로

본기를 제작하기 위한 회로는 크게 2가지로 구분되는데, 개발시에 사용되는 컴퓨터와 연결하는 샘플링회로와 실제로 분리되어 독립적으로 동작하는 회로로 나눌 수 있다. 전자는 A/D, D/A 콘버터, 필터, 앰프로 구성되며, 후자는 Z80을 CPU로 사용하는 원보드 컴퓨터와 확장메모리 , 저항래더를 사용한 D/A콘버터, 필터, 앰프, 키패드, LED 드라이버로 구성된다. 그러면 이제 이들 회로에 대한 설명과 제작방법을 각 부분별로 이야기해 보자.

### (1) 사용되는 부품

대부분의 IC는 TTL을 사용한다. 단 D/A 콘버터부분의 74HC374는 HC형을 쓰는 것이 좋다(LS로도 된다.) 그리 많은 부품이 사용되지는 않지만 C-MOS를 사용하거나 회로를 변경하고 싶지 않을 경우에는 TTL의 C-MOS 타입인 74HC형을 쓰면 전력소비가 작아지고, 논리레벨이 확실해진다.(LS형에서는 'H'일때 출력을 3V 정도 밖에 얻을 수 없으나 HC형의 경우는 4.99V가 보장되고 있다. D/A 콘버터에 HC형을 사용한 이유가 여기에 있다.) 사용되는 부품을 표 1에 보였다.

롬은 32K 바이트인 27C256이 좋다. 많은 양이 쓰이므로 C-MOS가 전력소

비면에서 유리하다. 액세스 스피드는 고려할 필요없이 아무 것이나 좋다.($V_{PF}$가 12.5V인 것, 21V인 것이 있으므로 자신이 사용하는 롬라이터로 지원이 가능한 것을 선택한다. 하지만 대부분의 롬라이터는 이 두 전압을 모두 지원한다.)

전원트랜스는 12~15V, 1A 정도의 것이면 좋다. A/D 콘버터는 가장 값싸고 입수하기 쉬운 ADC0809를 쓴다. ADC0808도 좋다. 그리고 D/A 콘버터에 사용하는 75k, 150k 저항은 1% 오차를 가진 것을 사용한다. 단일 전원으로 사용하기 위해서는 전용 D/A 콘버터를 사용하지 않고 저항래더를 사용하는데 1% 저항을 사용하면 큰 오차를 느끼지 못하기 때문에 쓸만한 D/A 콘버터가 된다. 여러 전원을 쉽게 얻을 수 있는 경우라면 전용의 D/A 콘버터를 쓰는 것이 좋다.

## (2) 샘플링회로의 제작

샘플링회로는 분리된 재생회로의 확장롬에 저장할 음성데이터를 샘플링하고 편집, 재생하기 위한 회로로서 IBM-PC에 인터페이스하여 사용한다. 샘플링회로에는 우선 음성신호를 컴퓨터에 저장 가능한 디지탈신호로 변환하기 위한 A/D 콘버터가 필요하며, 입력신호에 포함되어 있는 고주파성분과 노이즈를 제거하기 위한 필터, 저장된 신호를 재생하기 위한 D/A 콘버터와 앰프가 필요하다. 그 구성도는 **그림 2**와 같다.

**●A/D, D/A 콘버터**

IBM의 I/O채널에서 직접 디코딩한다. A/D 콘버터는 입수하기 쉽고, 사용하기 간편한 8비트 축차비교형 A/D 콘버터인 ADC0809를 사용한다. 이 IC의 구조와 기능에 대한 자세한 설명은 '음성출력시계'를 참고하기 바란다.

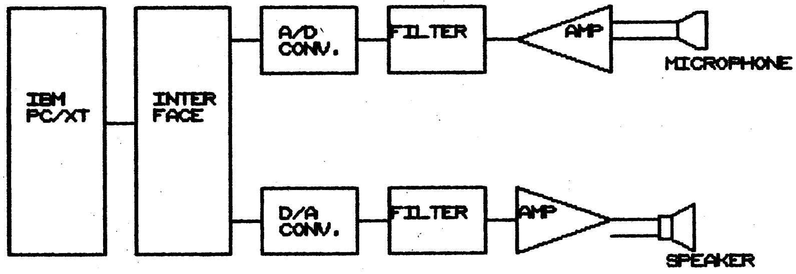
그리고 음성재생은 그렇게 정밀도를 요구하지 않으므로 D/A 콘버터는 간단히 저항래더를 사용한다. 이 경우의

〔표 1〕 부품표

| 명　칭 | 수 량 | 비　고 |
|---|---|---|
| 74154 | 1 | |
| 74LS04 | 1 | |
| 74LS32 | 1 | |
| 74LS93 | 1 | |
| 74LS138 | 2 | |
| 74LS139 | 2 | |
| 74LS125 | 1 | |
| 74HC374 | 1 | HC형이다. |
| ADC0809 | 1 | |
| Z80A | 1 | Z80A형을 사용 |
| 8255A-5 | 1 | |
| 6116 | 1 | |
| 2764 | 1 | |
| 27C256 | 5~8 | |
| IC 소켓 | | 사용하는 IC에 맞추어 구입 |
| IC용 만능 기판 | 3~4 | |
| 4MHz X-tal | 1 | |
| 1s1588 | 1 | 스위칭 다이오드면 모두 좋다. |
| LED | 17 | 적색의 고휘도형을 쓴다. |
| 푸시 스위치 | 17 | 소형 |
| 트랜스 | | 교류 12~15V 정도 |
| 브리지 다이오드 | 1 | 1~3a용이면 된다. |
| 휴즈 | 1 | 1a용의 통휴즈(홀더도 구입) |
| AC 코넥터 | 1 | |
| 7805 | 2 | 방열판도 구입한다. |
| C 220UF | 1 | 전해 |
| 10UF | 4 | |
| 47UF | 1 | |
| 100UF | 4 | |
| 220UF | 1 | |
| 0.1〔M〕 | 1 | 마일러 〔M〕, 앰프회로용 |
| 0.1〔C〕 | 4 | 세라믹 〔C〕 |
| 0.001〔M〕 | 1 | |
| 0.01〔M〕 | 3 | |
| 0.022〔M〕 | 6 | |
| 0.047〔M〕 | 6 | |
| 0.0022〔M〕 | 6 | |
| 앰프 키트 | 1 | 5.8 W 정도의 키트 |
| 스피커 | 2 | |
| 마이크 | 1 | 다이나믹 마이크 |
| 4558 | 4 | 호환품도 된다. |
| LM386 | 1 | |
| R10 | 1 | |
| 330 | 3 | |
| 1K | 2 | |
| 1.2k | 2 | |
| 3k | 3 | |
| 4.7k | 10 | |
| 6.8k | 1 | |
| 10k | 4 | |
| 13k | 4 | |
| 18k | 3 | |
| 27k | 3 | |

| 명　　칭 | 수 량 | 비　　고 |
|---|---|---|
| 75k(1%) | 14 | 1% 오차형을 사용한다. |
| 100k | 1 | |
| 150k(1%) | 16 | 1% 오차 |
| 4.7k 어레이 | 1 | 8 짜리를 사용한다. |
| 10k VR | 2 | 반고정형이면 된다. |
| 470k VR | 1 | |

〔그림 2〕 샘플링회로의 구성도




장점은 5V 단일 전원으로 간단히 구현할 수 있다는 점이다. 저항래더에 의한 D/A 변환을 이해하려면 포트에 연결된 저항래더의 한 엘리먼트(Element)가 2의 N승의 무게를 갖는다고 생각하면 된다. 그러므로 374의 출력포트의 비트 7에 물린 엘리먼트는 128, 비트 6은 64,...., 비트 0은 1의 무게를 가진다.

따라서 모든 출력이 'H'인 경우는 128+64+32+16+8+4+2+1=255가 되어 255가 출력된다. 반대로 모두 '0'인 경우는 당연히 0이 출력된다. 16진수가 &H55인 경우는 64+16+4+1=85=&H55 이 된다. 그러므로 입력된 값에 비례하는 아날로그 출력전압이 만들어지는 것이다.

이러한 래치(374)를 사용하는 이유는 D/A 콘버터로 출력된 데이터를 다음 데이터가 입력될 때까지 유지시켜 주기 위한 것이다. 회로는 그림 3과 같고, IC 만능기판에서 제작한다. 매우 간단한 회로이므로 쉽게 제작할 수 있을 것이다. 그리고 필터와 A/D, D/A 변환기판은 따로 분리하여 제작할 것을 권한다. 왜냐하면 이 회로는 일반목적의 A/D, D/A 콘버터로 사용가능하기 때문이다. 그리고 되도록 정확한 변환을 하기 위해서 주전원은 비교적 전압변동이 많은 +5V를 사용하지 않고 +12V로, 정전압 레귤레이터인 7805로 강압하여 사용하고 있는 점에 주의하기 바란다.

이렇게 만들어진 +5V와 IBM의 I/O 채널(확장 슬롯)에서 공급되는 +5V와 쇼트(단락)되지 않도록 주의하기 바란다. 필터회로와의 연결은 6핀 정도의 코넥터를 사용하면 된다. 또한 회로에서 볼 수 있는 데이터 풀업저항은 래치인 HC형 374의 입력전압레벨을 높여주기 위한 것이다.

**●필터 앰프회로**

OP 앰프로는 주로 많이 사용되는 4558을 사용한다. 음성신호를 샘플링하게 되면 입력과정에서 필요없는 잡음과 고주파성분이 포함되게 된다. 이러한 고주파 성분이 많아지면 샘플링한 음성에 많은 잡음이 포함된다. 그래서 A/D 변환 직전에 불필요한 신호를 제거해 주어야 하는데 이러한 역할을 필터가 해준다. 그리고 마이크를 통하여 입력되는 신호는 매우 미약하므로 증폭시켜 주어야 하고, D/A 변환된 신호도 스피커를 구동시키기에 충분한 신호로 저주파 증폭을 해주어야 한다.

그림 4를 보고 이를 제작하는데, 부품들이 대부분 소형이기 때문에 주의하여 제작한다. 출력부분의 10k 볼륨 이후의 회로는 저주파 앰프인데 기존의 앰프를 가지고 있다면 기존의 것을 이용해도 상관없다. 회로의 볼륨은 마이크쪽은 마이크의 입력레벨을 조절하기 위한 것이며, 출력쪽의 10k 볼륨은 스피커 출력조절용이므로 소리를 원하는 크기로 조절할 수 있다.

## (3) 재생회로

이 회로는 따로 독립하여 동작하게 되며, 입력된 선택신호에 해당하는 음성메시지를 출력해준다. 선택은 키패드를 이용하게 되며, Z80을 사용한 원보드 컴퓨터로 모든 것을 제어하게 된다. 원보드 컴퓨터는 키 입력을 감시하고 있다가 입력이 있는 경우 해당하는 음성신호를 재생하고, 해당하는 LED를 점등한 후 다시 키 입력을 감시하는 동작을 반복한다.

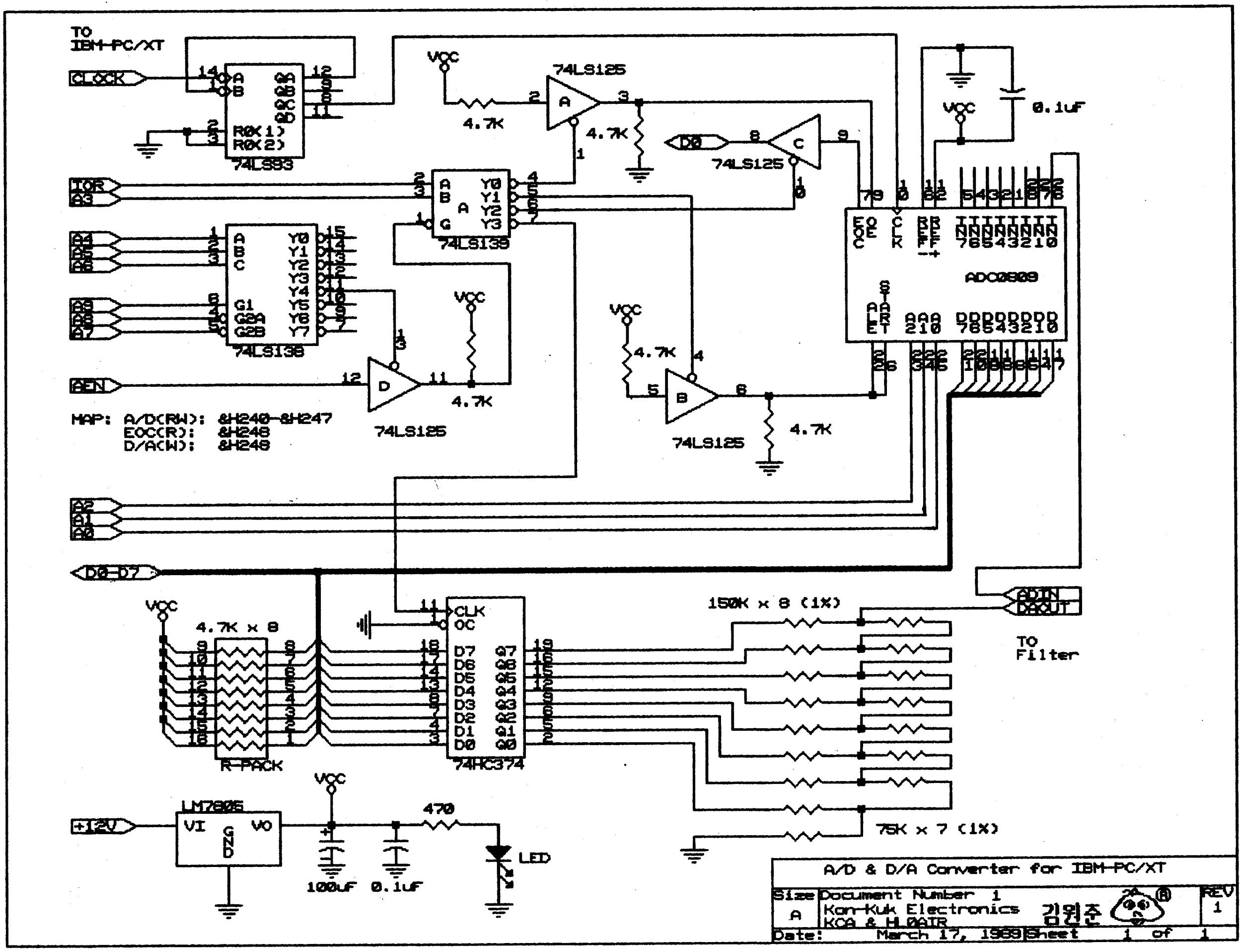
또한 고정된 회로가 아닌 컴퓨터 프로그램에 의한 동작이므로 제어 프로그램의 변경만으로 다양한 기능을 가지도록 할 수 있다. 이를 나름대로 활용해 보기 바란다. 그림 5는 본기의 구성을 나타낸다.

**●Z80 CPU회로**

이 회로는 다목적으로 사용 가능한 제어용 원보드 컴퓨터로 사용하기 간편하고 많이 사용되고 있는 Z80을 사용하였다. 이 회로 역시 매우 간단하게 제작하였으며, PPI로 24비트 입출력포트를 가진 8255를 사용하였다. 이 8255는 키패드 확장메모리 뱅크의 선택, LED 드라이브, D/A 변환에 사용된다. 이러한 원보드 컴퓨터는 다양하게 사용되므로 몇개 만들어 두면 제어용으로 사용할 때 무척 편리하다.

근래는 Z80이나 주변의 IC 가격이

〔그림 3〕 A/D, D/A 콘버터




매우 저렴해져서 어떤 회로를 설계 할 때, 좀 복잡한 기능이 필요하다면 주저하지 않고 제어용 원보드 컴퓨터를 사용하는 것이 단가나 기능, 확장면에서 모두 유리하다.

원칩 컴퓨터에는 CPU, 램, 롬, 입출력 포트가 대부분 하나의 칩안에 포함되어 있고 타이머, 시리얼 포트, A/D 변환기까지 내장된 것도 있다.

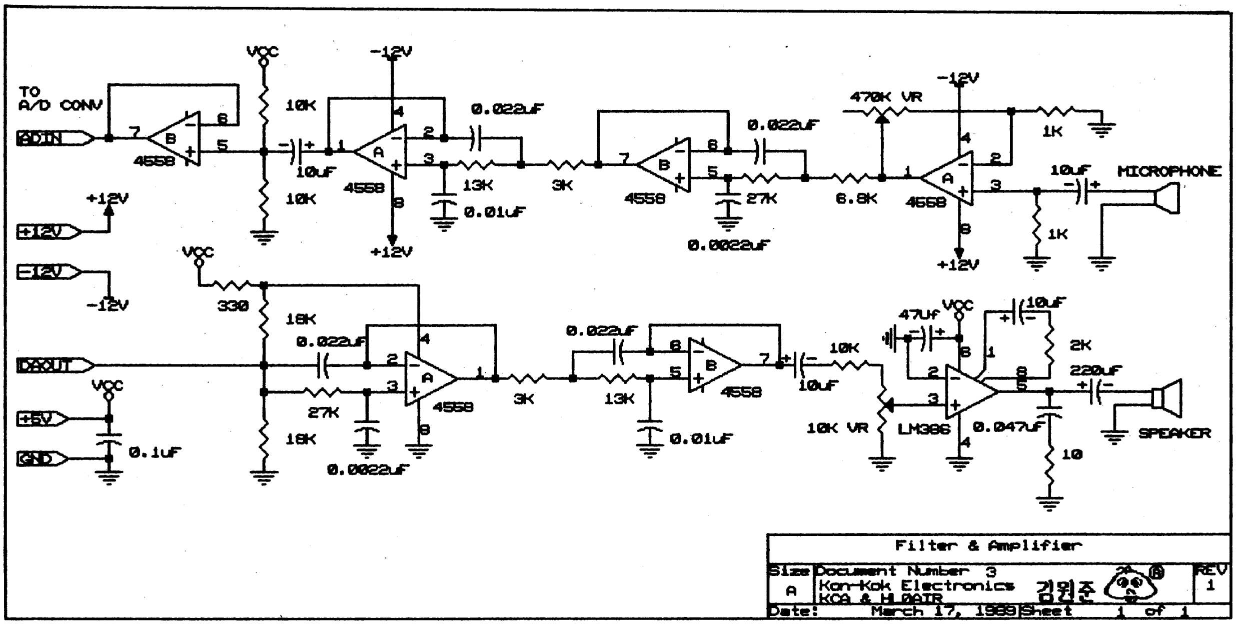
본기와 같이 복잡하지 않은 동작을 하는 회로는 원칩 컴퓨터를 사용하는 것이 유리하다. 특히 MCS-48 계열의 원칩 컴퓨터는 한글 번역판 매뉴얼도 나오고 있다. 이 계열은 좀 구식이지만 IBM의 키보드에도 사용되고 있고, 장거리 시외전화나 여러 콘트롤러에 많이 사용되고 있다. 그밖에 발전된 인텔의 MCS-51 계열이나 모토롤러의 68XX 시리즈, 자이로그의 Z80 등, 우수한 원칩이 많이 있다. 다른 일반 CPU를 사용해본 사람이라면 그리 어렵지 않게 원칩의 인스트럭션 세트를 숙지할 수 있고, 활용할 수 있을 것이다.

또한 IBM상에서 개발 가능한 크로스 어셈블러, 크로스 컴파일러나 개발 장비들도 많이 있다. 관심이 있는 사람은 사용해 보기 바란다. 이 회로의 메모리 배치는 **그림 6**과 같다. 램6116은 원보드 컴퓨터의 데이터저장을 하거나 스택으로 사용된다. 롬2764는 제어 프로그램이 들어간다.

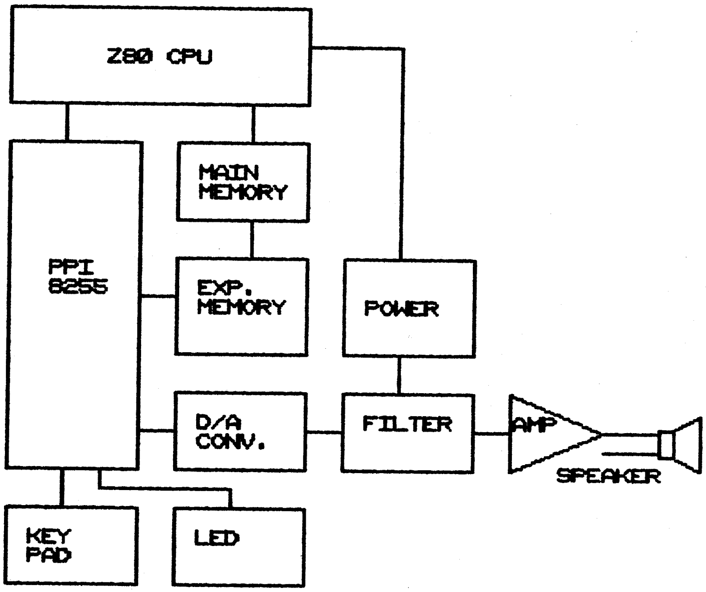
실제 제어 프로그램은 매우 짧아서 주메모리의 대부분이 비어 있게 된다. 그렇다고 더 작은 용량의 메모리를 사용할 필요는 없다. 롬의 경우는 이미 2732 이하의 것은 생산 중지되어 오히려 가격이 비쌀 것이다. 그리고 나중에 확장할 것을 고려하더라도 용량이 큰 것이 좋다. 이 주메모리는 Z80의 메모리 제어한계인 64K의 1/2영역의 앞쪽(0000-7FFFh)에 놓이고 확장메모리는 뒤쪽(8000h-FFFFh)에 놓인다.

그리고 확장롬은 뱅크방식을 사용하여 셀렉트할 수 있도록 되어 있다(그

〔그림 4〕 필터 앰프회로




〔그림 5〕 음성응답기의 구성도




림 6 참조). 처음에는 메모리부분과 메인보드를 분리해서 제작할 것을 예상하고 연결라인의 수를 줄이기 위하여 메모리의 어드레스를 카운터로 증가시키는 순차메모리를 설계하였으나 사용해 본 결과 확장메모리의 뒷부분에 배치된 데이터를 읽으려면 약간의 시간이 소모되며, 소프트웨어적인 조작이 많이 필요하므로 이와 같은 뱅크방식을 사용하였다.

이 방식의 장점은 I/O포트(8255)로 롬의 번호를 선택하여 주면 이후는 선택된 롬을 자신의 주메모리처럼 사용하게 되므로 자유롭게 액세스할 수 있다.

또한 디코더의 추가만으로 얼마든지 쉽게 확장 가능하다. 본기에는 256k바이트를 사용할 수 있으나, 실제로는 160k바이트 (32k×5)를 사용하였다. 이것만 해도 상당히 대용량이다. 아마추어에게는 이만한 양의 메모리를 사용하는 것도 무리가 될 것이다. 여하튼 자신이 필요한 만큼의 용량을 사용하면 된다. 이 원보드 컴퓨터에는 인터럽트를 사용하지 않았다. 필요한 경우에는 INT나 NMI단자를 빼내어 쓰면 되지만 본기의 목적에서는 인터럽트가 필요없다.

●PPI

입출력을 위하여 24비트 입출력 포트인 8255를 사용한다. 이것은 가장 많이 사용되고 있는 PPI(Programmable Peripheral Interface)이다. 이 IC를 이용하여 키패드와 LED, 확장메모리, D/A 콘버터를 모두 제어한다. Z80의 경우는 외부 디바이스의 선택에 IORQ를 사용하는데 I/O용 디바이스는 이

회로에서 PPI하나 뿐이므로 어드레스 디코딩할 필요없이 IORQ만 선택되면 무조건 8255가 선택되도록 하였다.

8255에 대한 자세한 설명은 '음성출력시계'에서 자세히 언급되어 있으며, 하드웨어를 취급한 대부분의 서적에서 8255에 관한 자료는 쉽게 구할 수 있을 것이다. Z80과 PPI가 포함된 회로는 **그림 7**에 있다.

**●키패드**

선택신호를 입력하기 위한 키보드이다. 뒤의 소프트웨어에서 다시 설명하겠지만 스캐닝방식으로 키를 입력받는다. 포트를 하나 더 연결하면 이 상태로 회로의 변경없이 16개 키를 추가할 수 있고 2개의 포트를 더 사용하면 모두 32개까지 키를 입력 받을 수 있다. 하지만 음성신호는 대용량의 메모리를 필요로 함으로 아주 절약하여 사용하지 않으면 16개 이상의 메시지를 저장하기는 어려울 것이다.

**●LED**

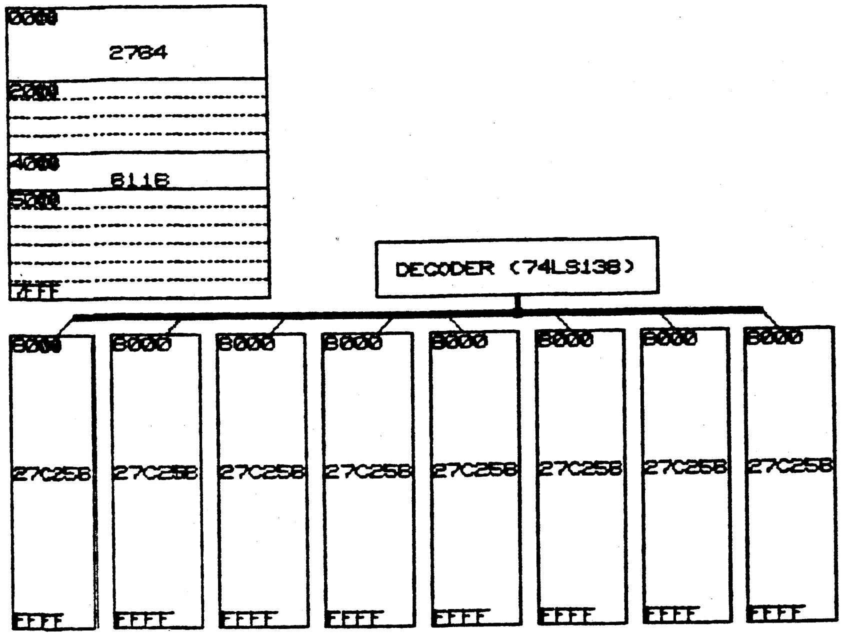
반드시 필요한 것은 아니지만 어느 메시지가 선택되어 출력되고 있는지를 확인하기 위하여 LED를 달아준다. 이것은 모두 16개인데, 4×16 디코더인 74154를 이용하여 간단히 구동하였다. 회로를 보면 알 수 있듯이 74154의 출력에 버퍼나 트랜지스터를 달지않고 직접 LED를 구동하고 있다. 하지만 전혀 이상은 없고 74154의 출력은 LED를 드라이브하기에 충분했다. 그리고 74154에 입력되는 선택신호는 8255를 사용한다. 이 기능이 필요하지 않은 경우는 생략해도 관계없다.

이때 남게되는 8255의 포트는 다른 목적으로 활용할 수 있다. 특히 선택신호의 입력에 키를 사용하지 않고 외부기기로부터 선택을 받으려는 경우는 외부기기에 동작중과 동작종료를 알려주어야 하는데 이러한 경우 LED용으로 할당된 포트를 사용하면 좋다.

**●확장메모리의 선택**

확장메모리는 27C256를 사용하는데 이 롬은 한개가 32K바이트의 용량을

〔그림 6〕 메모리 배치도




가진다.(만일 27512를 사용하고자 하는 경우에는 A15를 사용하여 롬을 반으로 나누어 사용하면 된다.) 그러므로 원보드 컴퓨터의 빈 메모리영역을 꽉 채우게 된다. 그래서 한번에 한개의 롬만을 선택하면 되는데, 3×8 디코더인 74LS138을 사용하여 8개까지 롬을 선택할 수 있도록 하고 있다. LED에서의 경우와 마찬가지로 74LS138에 입력되는 선택신호는 8255로부터 나온다.

많은 양의 롬을 사용하는 데도 버퍼를 사용하지 않고 있다. 물론 한번에 선택되는 롬은 한개 뿐이므로 문제는 없지만 안정된 동작을 하기 위해 데이터라인 정도는 버퍼를 달아 주는 것이 좋을 것이다. 8개의 롬에는 RD신호를 연결하지 않는데 그것은 항상 읽기전용(Read Only)이기 때문이다. 그래서 디코더에서 RD신호가 'L'일때만 확장롬이 선택되도록 하고 있다.

회로에서 보는 바와 같이 데이터나 어드레스 버스를 모두 연결하는 것은 엄청난 작업이다. 깔끔하고 편리하게 작업하기를 원하는 사람은 세운상가에서 확장메모리용으로 다수의 롬을 꽂을 수 있고, 어드레스와 데이터가 모두 연결되어 있는 작은 기판을 구할 수 있을 것이다. 이 기판을 사용하면 전체 작업량의 반(?)을 줄일 수 있을 것이다. 회로는 **그림 8**과 같다.

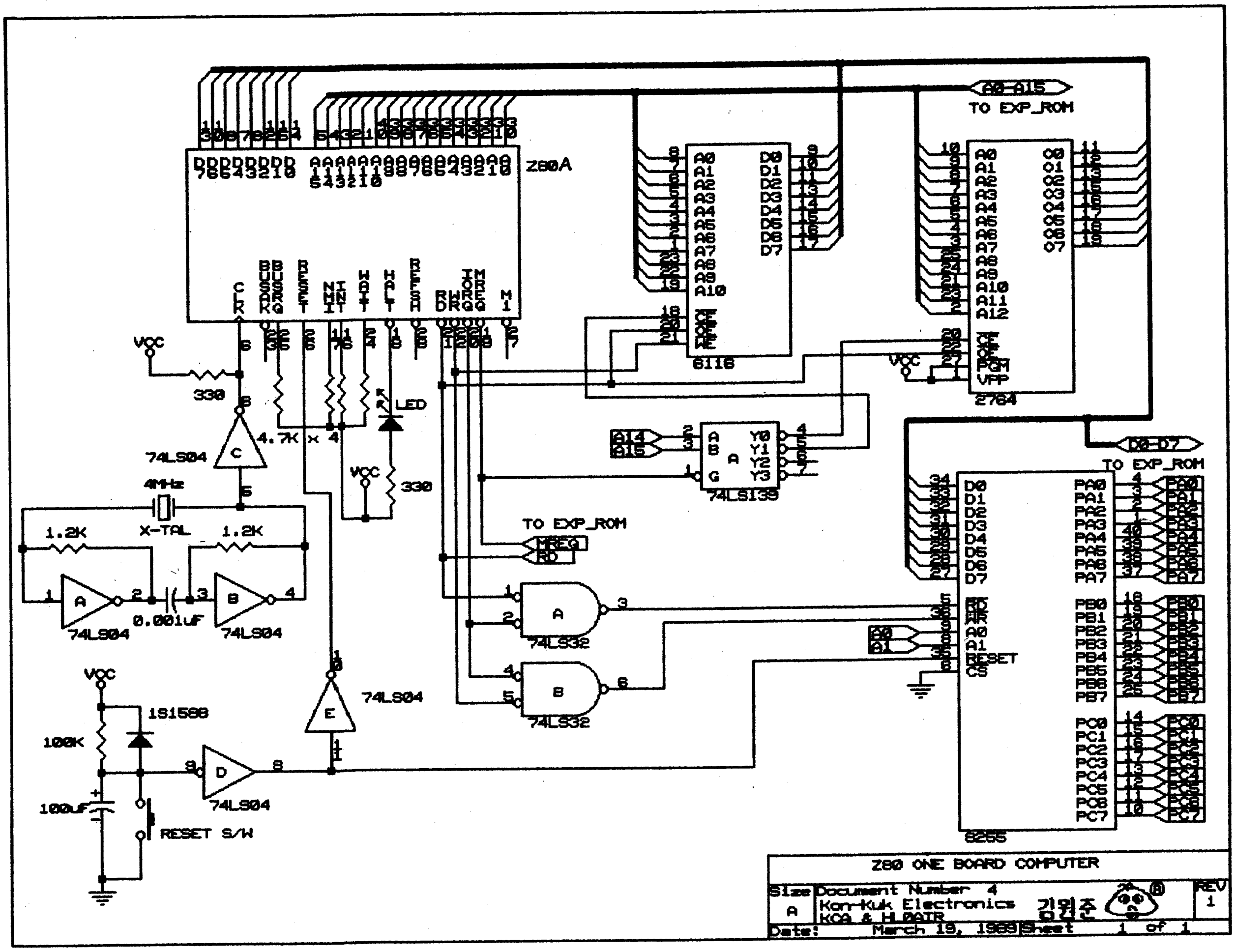
**●D/A콘버터**

독립된 재생기의 경우는 신호의 샘플링을 할 필요없이 재생만 하면 되므로 A/D 콘버터는 필요없고 D/A 콘버터만 있으면 된다. 그래서 간단히 여기서도 저항래더(사다리 회로)를 사용하여 제작한다. 이 D/A 콘버터는 5V 단일 전원으로 동작한다. 8255가 래치기능을 가지므로 시그널 홀드용의 래치는 필요 없어지며 8255의 출력레벨도 5V에 훨씬 못미치지만 음성출력용의 D/A 변환에서 이러한 낮은 출력은 스피커 출력을 조금 감소시킬 뿐이다. 그러므로 앰프의 볼륨을 조금 높여주면 충분히 보상(?) 받을 수 있을 것이다.

**●필터 앰프**

앞의 샘플링회로 필터앰프와 같다. 단 이 경우는 분리되어 동작하므로 좀 큰 출력이 필요할 것이다. 그래서 386

〔그림 7〕 Z80 원보드 컴퓨터




을 사용한 앰프대신 좀 고출력의 앰프를 사용하여야 한다. 그런데 이러한 앰프를 직접 제작하는 것보다는 시판되고 있는 키트를 사용하는 것이 유리하다. 가격도 1500~2000원선이면 6~10 W정도의 쓸만한 앰프 키트를 구할 수 있다. 이 경우는 전원이 문제인데, 다른 IC용의 5V와 같이 쓰는 것은 좋지 않으므로 전원의 정류기를 거쳐 나온 12V 정도의 전원을 사용하면 좋을 것이다( 그림 9 참조).

●전원

컴퓨터로부터 전원을 얻을 수 없으므로 독립된 전원이 필요하다. 그래서 그림 9와 같은 전원을 제작한다. 7805에는 대부분 많은 열이 발생하므로 반드시 방열판을 달아준다. 트랜스에서 강압되어 나온 교류를 정류하여 앰프용의 전원으로 사용하고 다시 7805(정전압 레귤레이터)로 +5V를 주전원으로 공급한다.

필터회로도 단일 전원으로 동작하도록 되어 있으므로 그 밖의 전원은 필요없다. 완성된 음성응답기의 예를 사진으로 보였다. 사진 1은 확장롬 기판을 연결한 것이고, 사진 2는 확장롬 기판을 분리한 모습이다. 필자는 기판간의 연결에 케이블을 사용하지 않고 두 기판의 마주 보는 면에 암수 코넥터를 사용하여 위에서 꽂을 수 있도록 하였다. 이렇게 하면 깨끗하고 간편하게 제작할 수 있다.

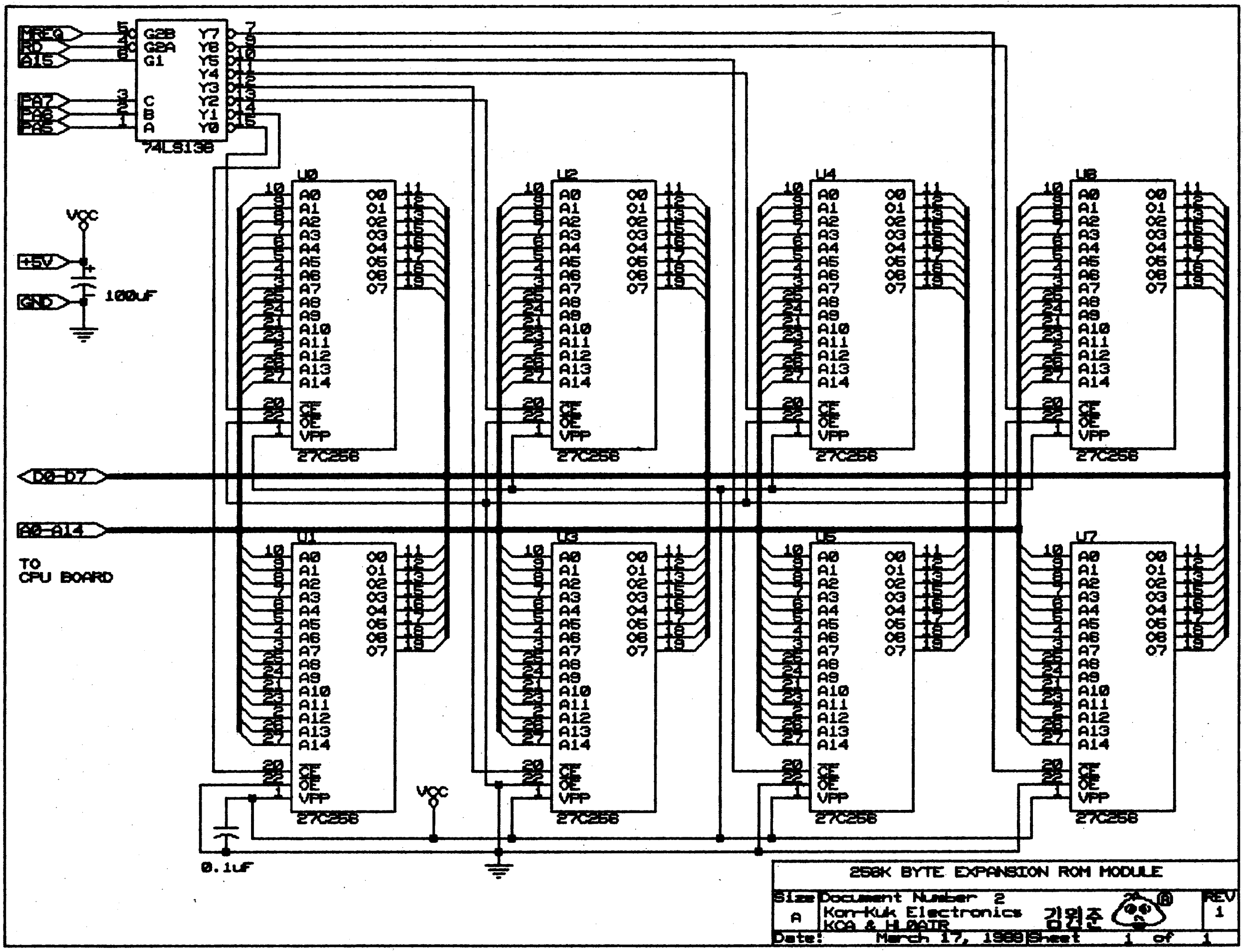
## 소프트웨어 및 사용방법

이상없이 만들어졌는지 확인한 후 동작시켜 본다.

### (1) 샘플링 회로

우선 제어 프로그램에 대하여 알아보자. 샘플링을 하기 위한 프로그램은 리스트 1과 같다. 이러한 음성처리는 리얼타임에 가까운 샘플링이 필요하고, 고속을 요구하기 때문에 고급언어를 사용하기가 어려워 어셈블리어를 사용

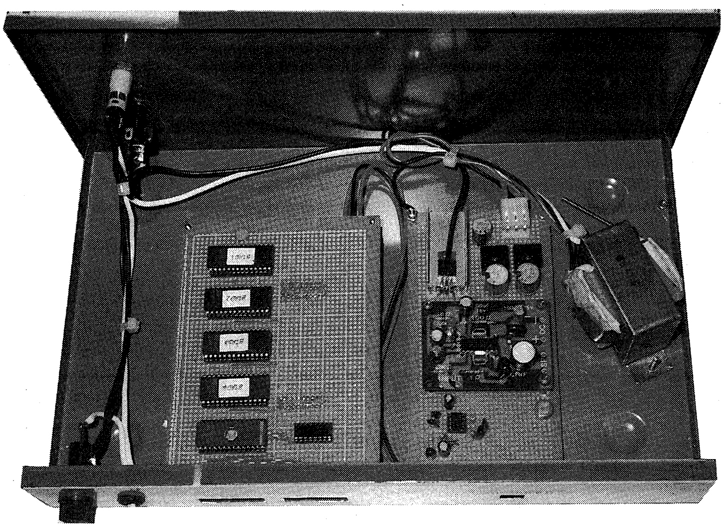
〔그림 8〕 확장 메모리(롬)



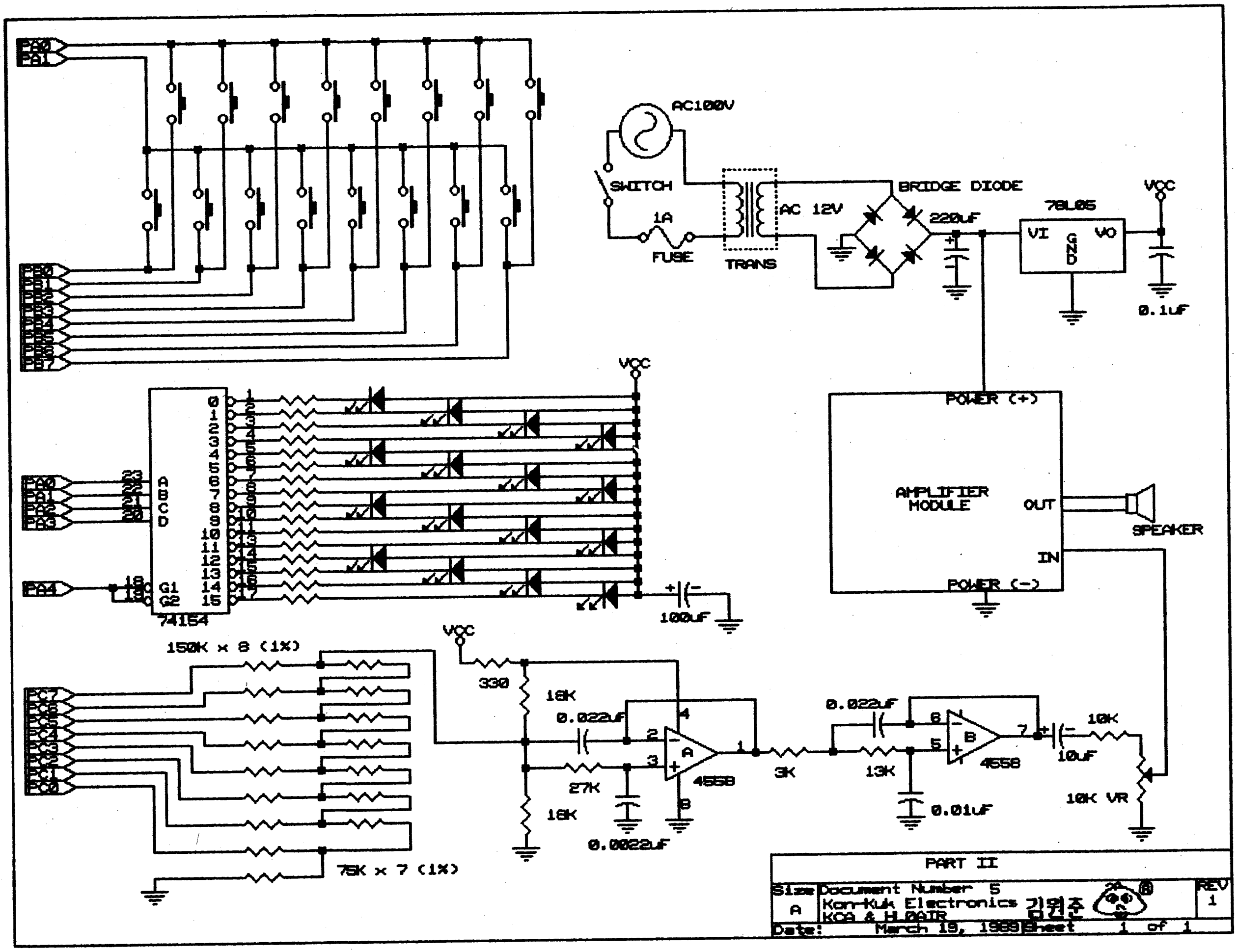

하였다. 제대로 하기 위해서는 CPU의 OP코드의 클럭소모 갯수까지 알아야 하는데, 그 정도의 정밀도는 필요없다. 하지만 알고 싶은 독자는 한번 계산해 보기 바란다. 일반 서적에서는 그러한 정보를 구할 수 없고, Lewis C.Egger-brecht 의 'Interfacing To The IBM Personal Computer'에서 구할 수 있다. 번역판도 있다. 단 국내 IBM 기종은 클럭 주파수가 다양하므로 독자들이 갖고 있는 기종의 클럭주파수를 고려해야 할 것이다.

리스트 1의 프로그램은 3가지의 동작을 한다. 우선 음성신호를 입력 받아서 메모리에 저장하는 기능, 저장된 내

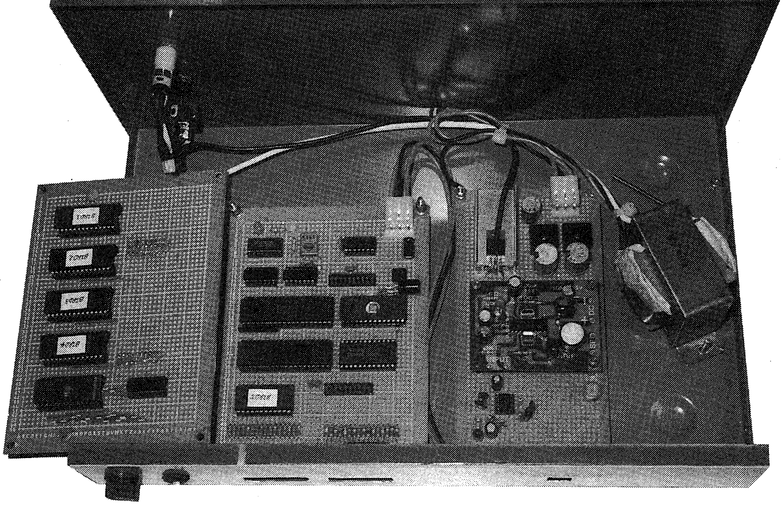
〔사진 1〕 확장 롬 기판을 연결한 모습



〔그림 9〕 키패드, LED, D/A 콘버터, 전원 회로도




〔사진 2〕 확장 롬 기판을 분리한 모습



용을 출력하여 확인하는 기능, 볼륨의 레벨을 조절하기 위하여 마이크로 입력되는 신호를 그대로 다시 출력해 주는 기능이다. 리스트 1을 적당한 에디터로 입력한 후 컴파일하여 sam.com이라는 화일로 만든다. 다음에는 디버그를 이용하여 데이터 영역을 확보해 주어야 한다.

이렇게 하는 이유는 샘플링을 한후 데이터 화일을 따로 만들지 않고 주프로그램의 뒷부분에 데이터를 로드한 후 함께 저장하는 방식을 사용하기 위함이다. 그 과정을 프로세스 1에 보였다. 이 프로그램에서는 1000H~8FFFH의 32K 바이트가 데이터 저장용으로

사용된다. 그러므로 롬에 저장할 때는 롬라이터로 이 부분을 롬에 라이트하면 된다.

이제는 부분적으로 살펴보자. 실제 동작도 DOS상태가 아닌 디버그상에서 실행시킨다. 우선 필터와 마이크, 스피커를 연결한 후 A/D 콘버터와 필터 기판을 코넥터로 연결한다. 그 다음에는 IBM의 확장슬롯에 주의하여 꽂는

〔리스트 1〕 샘플링 프로그램

```
;************************************************************;
;        SMAPLING & ANALOG/DIGITAL CONVERSION PROGRAM        ;
;    BY KIM WEON JUN [KON_KUK ELECTRONICS, HMAIL_ID:KCA]     ;
;************************************************************;


conv_ad   equ     0240h
da_conv   equ     0248h
start_me  equ     1000h          ; LOMEM
len_data  equ     8000h          ; 데이터 길이
cycle_dl  equ     0038h          ; 지연시간

;-------------------------------------------------------------

code      segment
          assume  cs:code,ds:code

;-------------------------------------------------------------

          org     0100h

start:
          push    cs             ; 세그먼트 정의
          pop     ds

          cli                    ; 인터럽트 디스에이블

;-------------------------------------------------------------

          mov     bx,start_me    ; 스타트 메모리 로드
          mov     cx,len_data
sam1:     xor     al,al          ; 변환 시작
          mov     dx,conv_ad
          out     dx,al
          call    delay          ; 대기
          mov     dx,conv_ad
          in      al,dx          ; I/O로 부터 샘플링 데이터 로드
          mov     [bx],al        ; 데이터 저장
          inc     bx
          loop    sam1           ; 계속
          jmp     eop

;-------------------------------------------------------------

          mov     bx,start_me    ; 스타트 메모리 로드
          mov     cx,len_data
sam2:     mov     dx,da_conv
          mov     al,[bx]
          out     dx,al          ; 음성출력
          call    delay          ; 대기
          nop
          nop
          inc     bx
          loop    sam2           ; 계속
          jmp     eop

;-------------------------------------------------------------

          mov     bx,start_me    ; 스타스 메모리 로드
          mov     cx,len_data
sam3:     xor     al,al          ; 변환 스타트
          mov     dx,conv_ad
          out     dx,al
          call    delay          ; 대기
          mov     dx,conv_ad
          in      al,dx          ; I/O로 부터 샘플링 데이터 로드
          mov     dx,da_conv
          out     dx,al          ; 음성출력
          inc     bx
          loop    sam3           ; 계속

;-------------------------------------------------------------

eop:      sti                    ; 인터럽트 인에블
          int     20h            ; DOS로 빠져 나감

;-------------------------------------------------------------

delay     proc    near           ; 지연 서브루틴
          push    cx
          mov     cx,cycle_dl
timer:    loop    timer
          pop     cx
          ret
delay     endp

;-------------------------------------------------------------

code      ends
          end     start          ; 프로그램 종료
```

다. 그리고

A>debug sam.com

으로 디버거로 프로그램을 로드한다.
다음에는 샘플링을 하는데

-g=100 (시작 번지)

을 하면 마이크로 입력되는 데이터가 메모리에 저장된다. 약 5초후 프로그램 종료신호가 나올 것이다.
다음은 재생을 해보자.

-g=XXXX (리스트1의 A로 표시된 부분의 실제번지 -디버그로 확인)

를 하면 지금까지 입력하였던 내용이 스피커를 통하여 흘러나올 것이다.
그리고 음량을 조절하기를 원한다면

-g=YYYY (YYYY:리스트1의 B로 표시된 부분의 실제번지)

를 입력한다. 이것은 일정한 시간동안 음성이 입력되는 것과 동시에 출력된다. 흡사 앰프와 같다. 이 루틴을 실행시켜서 적당한 음량을 찾고 음량조절은 입출력단에 있는 볼륨을 사용한다.
이상없이 저장된다면 자신이 원하는 메시지를 저장한 프로그램을 적당한 이름으로 저장해둔다. 리스트 1의 재생에 해당하는 sam2 : 루틴의 start_me, len_data는 각각 재생할 메모리의 시작, 길이를 나타내므로 이 값을 적당히 조절하면 원하는 부분만을 재생할 수 있다. 이렇게 원하는 부분만을 골라서 편집을 하여 원하는 데이터가 메모리 어느 번지부터 저장되어 있고 길이는 얼마나 되는지 기록해 둔다.

●영위법에 의한 저장과 재생

이제 음성을 저장하고 재생해 보았다. 성공을 했다면 느낄 수 있었겠지만 기대한 것 이상의 음질을 얻을 수 있었을 것이다. 8비트의 해상도를 사용해도 음성을 샘플링하는 데는 큰 지장이 없다.
그러면 여기서 한가지 응용을 해 보도록 하자. 보통 프로그램을 작성할 때

〔프로세스 1〕 샘플링 프로그램 컴파일

```
C:\ASM\MASM>
C:\ASM\MASM>
C:\ASM\MASM>masm sam;                           ←프로그램을 컴파일한다
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

  51686 + 347146 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\ASM\MASM>link sam;                           ←링크한다

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

LINK : warning L4021: no stack segment

C:\ASM\MASM>exe2bin sam sam.com                 ←.COM 화일로 만든다

C:\ASM\MASM>dir sam.*                           ←확인

 Volume in drive C is K. Weon-Jun
 Directory of  C:\ASM\MASM

SAM      ASM     3932   3-20-89   7:09a
SAM      OBJ      171   3-20-89   7:13a
SAM      EXE      858   3-20-89   7:13a
SAM      COM       90   3-20-89   7:13a     ←만들어졌다
SAM      BAK     3932   3-20-89   7:06a
        5 File(s)   1105920 bytes free

C:\ASM\MASM>debug sam.com                       ←디버그로 들어간다
-rcx
CX 005A                                         ←데이터영역을 만든다
:8f00
-w                                              ←Write
Writing 8F00 bytes
-q                                              ←Quit

C:\ASM\MASM>dir sam.com                         ←확인

 Volume in drive C is K. Weon-Jun
 Directory of  C:\ASM\MASM

SAM      COM    36608   3-20-89   7:13a  ←완성
        1 File(s)   1071104 bytes free

C:\ASM\MASM>
C:\ASM\MASM>
C:\ASM\MASM>
```

는 화면에 메시지를 표시하여 진행하는데, 프로그램이 음성으로 안내를 해 준다면 재미있을 것이다. APPLE Ⅱ 컴퓨터용 프로그램 중에는 sam이라는 것이 있는데, 이것은 음성을 음소별로 조합하여 합성음을 알려준다. IBM에도 speech라는 sam과 같은 프로그램이 있다. 하지만 이들은 영어를 사용하는 나라에서 만들어졌기 때문에 우리

〔리스트 2〕 영위법에 의한 프로그램

```
;*************************************************************;
;       SMAPLING & ANALOG/DIGITAL CONVERSION PROGRAM          ;
;    BY KIM WEON JUN [KON_KUK ELECTRONICS, HMAIL_ID:KCA]      ;
;*************************************************************;

conv_ad   equ     0240h              ; A/D콘버터, I/O 맵
start_me  equ     0200h              ; LOMEM 데이터 메모리
len_data  equ     1e00h              ; 데이터 길이
cycle_dl  equ     0035h              ; 지연시간

;-------------------------------------------------------------

          org     0100h

code      segment
assume    cs:code,ds:code     ; 코드 세그먼트 정의

;-------------------------------------------------------------

start:
          mov     ax,cs              ; 데이터 세그먼트 정의
          mov     ds,ax
          cli                        ; 인터럽트 디스에이블

;-------------------------------------------------------------

          mov     bx,start_me
          mov     cx,len_data
daconv:   mov     dh,[bx]
          push    cx
          mov     cx,08h
rotate:   mov     ah,dh
          and     ah,80h
          rol     ah,1
          rol     ah,1
          in      al,61h
          and     al,0fdh
          or      al,ah
          out     61h,al
          call    delay
          nop
          nop
          rol     dh,1
          loop    rotate
          pop     cx
          inc     bx
          loop    daconv
          jmp     eop

;-------------------------------------------------------------

          mov     bx,start_me        ; 스타트 메모리 로드
          mov     cx,len_data
continue: push    cx
          xor     ah,ah
          mov     cx,08h
sam1:     xor     al,al              ; 변환 스타트
          mov     dx,conv_ad
          out     dx,al
          call    delay              ; 대기
          mov     dx,conv_ad
          in      al,dx              ; 샘플링 데이터
          and     al,80h
          or      ah,al
          rol     ah,1
          loop    sam1
          pop     cx
          mov     [bx],ah            ; 데이터 저장
          inc     bx
          loop    continue           ; 계속

;-------------------------------------------------------------

eop:      sti                        ; 인터럽트 인에이블
          int     20h                ; DOS로 빠져 나감

;-------------------------------------------------------------

delay     proc    near               ; 지연 서브루틴
          push    cx
          mov     cx,cycle_dl
timer:    loop    timer
          pop     cx
          ret
delay     endp

;-------------------------------------------------------------

code      ends                       ; 세그먼트 종료
          end     start              ; 중지
```

말을 합성하게 되면 매우 어색한 소리가 난다. 그러면 지금 만든 회로를 이용할 수는 없을까?

물론 가능하다. 하지만 한가지 문제가 되는 것은 D/A 콘버터가 필요하다는 것이다. 프로그램에서 하드웨어적인 추가없이 음성이 나오도록 하자면 내장된 스피커를 사용하여야 한다. 그런데 IBM의 내장 스피커는 ON/OFF의 기능 밖에 없어서 주파수의 변화는 가능하지만 진폭은 자유롭게 변환시킬 수 없다. 게다가 엄청난 메모리를 소모하는 것도 문제가 된다. 그래서 음질은 상당히 떨어지지만, 영위법을 사용하면 그래도 알아들을 만한 소리를 들을 수 있고, 하드웨어적인 추가도 없으며, 메모리도 현재의 1/8만이 소모된다.

방법은 음성신호를 1비트로 샘플링하는 것이다. 무신호시의 입력값을 기준으로 신호의 진폭이 (−)일때는 0, (+)일때는 1로 하여 저장한다. 그러면 1바이트에 8개의 신호를 저장할 수 있다. 짐작할 수 있듯이 이 방법은 음질이 상당히 떨어진다. 하지만 재미는 있으므로 한번쯤 시도해 보기 바란다. 이 제어 프로그램은 **리스트 2**와 같다. 이 프로그램을 입력시켜 **프로세스 2**와 같이 컴파일한 후에 디버거로 데이터 영역을 확보한다. 그리고

```
-g=
```

로 실행시켜서 음성을 저장한다. 저장이 끝나면 프로그램을

```
-w
```

로 다시 저장한 후 DOS로 나간다. 이후는 그 프로그램을 DOS상에서 실행시키면 저장된 음성이 내장 스피커를 통하여 흘러나온다.

**●4비트 저장방식**

사용해 보면 느끼겠지만 메모리의 소모가 너무 많을 것이다. 그래서 이를 보완하기 위한 방법 중의 하나로 4비트 저장방식을 생각해 보자. 음성신호는 인간의 성대를 울려서 나오는 소리이므로 급격하게 변하지는 않는다. 그러므로 샘플링속도만 충분히 빠르다면 지난번의 샘플링 데이터와 현재의 데이터의 차를 저장하는 방식을 사용할 수 있다. 이 차이의 값은 대부분 16이내에서 변하므로 4비트로 저장 가능하다. 그러므로 메모리를 1/2로 줄일 수

〔프로세스 2〕 영위법을 사용한 프로그램의 컴파일

```
C:\ASM\MASM>
C:\ASM\MASM>masm voice2                          ←컴파일
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.
Object filename [voice2.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51666 + 345726 Bytes symbol space free

     0 Warning Errors
     0 Severe  Errors
C:\ASM\MASM>link voice2                          ←링크

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [VOICE2.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment       ←이 주의는 무시해도 된다

C:\ASM\MASM>exe2bin voice2 voice2.com

C:\ASM\MASM>debug voice2.com                 ←디버그로 읽는다
-rcx
CX 0064                                      ←데이터영역 확보
:1f00
-w                                           ←Write
Writing 1F00 bytes
-q                                           ←Quit

C:\ASM\MASM>dir voice2.*                     ←확인

 Volume in drive C is K. Weon-Jun
 Directory of  C:\ASM\MASM

VOICE2   BAK     3084   1-01-80  12:55a
VOICE2   OBJ      177   3-21-89  12:13p
VOICE2   EXE      612   3-21-89  12:13p
VOICE2   COM     7936   3-21-89  12:13p     ←완성
VOICE2   ASM     3084   3-21-89  12:11p
        5 File(s)   1224704 bytes free

C:\ASM\MASM>
C:\ASM\MASM>
```

있다.

하지만 이 방법을 사용할 때는 고해상도의 A/D 콘버터를 사용하기를 바란다. 이유는 데이터의 변화는 (+)와 (−)로 모두 변화하므로 4비트 중 첫 비트는 부호 플래그로 사용되어야 하며 −8~+7의 변화만을 감지할 수 있다. 그런데 이중 한번이라도 이 범위를 벗어나게 되면 이어서 많은 오차를 발생하게 된다.

특히 A/D 콘버터의 최하위비트는 오차가 심하므로 더욱 오차의 발생소지가 높아진다. 따라서 12비트 A/D 콘버터인 ADC1211 등을 사용하여, 하위의 4비트를 무시하고 사용한다면 충분히 사용 가능할 것이다. ADC0809로도 소프트웨어적인 보상을 잘 해준다면 사용할 수는 있지만 확실한 방법을 선택하기로 했다.

## (2) 음성 응답기

이제는 재생기의 프로그램에 대하여 알아 보자. 본기에는 Z80을 CPU로 사용하므로 IBM상에서 MASM 등으로 개발할 수 없다. 그래서 Z80 크로스 어셈블러인 XASMZ80.COM을 사용하여 IBM상에서 개발한다. 이 크로스 어셈블러는 세운상가 등에서 쉽게 구할 수 있을 것이다. 사용법은 Z80을 사용해본 사람에게는 아무런 어려움이 없을 것이다.

또한 데모용 화일이 들어 있으므로 그것만으로도 충분히 사용법을 익힐 수 있다. 표준 니모닉을 사용하고 타기종의 어셈블러와 의사명령도 거의 같다. Z80을 사용하는 8비트 컴퓨터인 SPC-1000, MSX 등이나 APPLE Ⅱ의 CP/M상에서 개발해도 좋다. 단, 어느 경우나 롬라이터를 필요로 하므로 롬라이터가 사용 가능한 기종을 선택하는 것이 좋을 것이다.

여기서는 IBM의 크로스 어셈블러를 사용하는 것을 기준으로 하겠다. 이 크로스 어셈블러는 IBM상에서 짜여진 Z80 어셈블리 프로그램을 컴파일하여 Z80상에서 실행될 수 있는 기계어코드로 만들어 .HEX라는 확장명을 갖는 화일로 만들어 준다. 그러므로 IBM상에서 직접 실행시킬 수는 없고, 롬라이터나 에뮬레이터 등으로, Z80으로 프로그램을 넘겨서 실행시켜야 한다.

필자의 경우는 IBM에 Z80을 탑재한 원보드 컴퓨터를 꽂아서 이 Z80을 IBM으로 제어할 수 있도록 하고 프로그램의 다운, 업 로드를 DMA 방식으로 하게 하고 있다. 이렇게 하면 IBM상에서 완전하게 Z80 프로그램을 개발하고 실행, 수정할 수 있다.

그럼 프로그램을 살펴 보기로 하자. 프로그램의 앞부분에는 음성 데이터의 테이블이 들어 간다. 확장롬은 8개까지 사용하므로 원하는 데이터를 골라 내기 위해서는 롬의 번호와, 시작번지, 길이 등의 데이터가 필요하다. 또 몇번 반복하여 출력하고자 하는 경우를 고려하여 반복 회수도 지정할 수 있도록 한다. 그리고 재생속도도 조절할 수 있으면 좋다. 이렇게 5가지의 정보를 기록해둔다.

프로그램은 선택 입력이 있을 때 해당하는 데이터의 테이블을 참조하여 어느 롬의 어느 번지부터, 얼마나 많은 데이터를, 어떤 속도로, 몇번이나 출력할 것인가를 알아 낸후 재생하게 된다. 프로그램은 **리스트 3**에 있고, 프로그램을 크로스 어셈블러로 컴파일하는 과정을 **프로세스 3**에 보인다. 프로그램은 IBM의 샘플링회로와 마찬가지의 동작을 수행하므로 별다른 설명이 필요없을 것이다. 프로그램을 자세히 보면 키보드와 LED, D/A 콘버터를 8255하나만으로 모두 제어할 수 있도록 신경을 쓰고 있으므로 분석해 보기 바란다.

〔리스트 3〕 음성응답기 제어 프로그램(Z80 크로스 어셈블러)

```
AVOCET SYSTEMS Z80 ASSEMBLER -  VERSION 1.04M  SERIAL #00329

SOURCE FILE NAME: BIRD_COM.ASM


                 ;-----------------------------------------------------------------
                 ;
                 ;   Bird-Com V1.0 [1/17] by Kim Weon Jun
                 ;   Kon-Kuk Univ. Electronics #871369  [K.C.A & HLOAIR]  Tel.463-1387
                 ;
                 ;-----------------------------------------------------------------
                 ; 등식
                 ;
0000             pa      equ  0h          ; 8255 포트 A
0001             pb      equ  1h          ; 8255 포트 B
0002             pc      equ  2h          ; 8255 포트 C
0003             ctrl    equ  3h          ; 8255 콘트롤 포트
4400             keybuf  equ  4400h       ; 키 버퍼
4800             stack   equ  4800h       ; 주메모리 맵 : &h0000-&h1fff(27c64)
0000                                      ;                   &h4000-&h47ff(6116)
0000                     org  0000h       ; I/O 셀렉트 롬: &h8000-&hffff(27c256)*8
0000                                      ;
0000 C30001              jp   start       ;

                 ;-----------------------------------------------------------------
                 ; 데이터 테이블[Rom0]
```

```
                 ;
0003 00          bird_0: db   0          ; Ext.Rom 번호  1번째 데이터
0004 0080                dw   08000h     ; 데이터 스타트 포인터
0006 001C                dw   01c00h     ; 데이터 길이
0008 0532                db   5,32h      ; 반복, 지연시간

000A 00          bird_1: db   0          ; 2번째 데이터
000B 009C                dw   09c00h
000D 001F                dw   01f00h
000F 0333                db   3,33h

0011 00          bird_2: db   0          ; 3번째 데이터
0012 00BB                dw   0bb00h
0014 002D                dw   02d00h
0016 0234                db   2,34h

0018 00          bird_3: db   0          ; 4번째 데이터
0019 00E8                dw   0e800h
001B 0018                dw   01800h
001D 0342                db   3,42h

                 ;-----------------------------------------------
                 ;
                 ; 데이터 테이블[Rom1]
                 ;
001F 01          bird_4: db   1          ; 5번째 데이터
0020 0080                dw   08000h
0022 0018                dw   01800h
0024 0335                db   3,35h

0026 01          bird_5: db   1          ; 6번째 데이터
0027 009D                dw   09d00h
0029 001C                dw   01c00h
002B 0335                db   3,35h

002D 01          bird_6: db   1          ; 7번째 데이터
002E 00B9                dw   0b900h
0030 001A                dw   01a00h
0032 0335                db   3,35h

0034 01          bird_7: db   1          ; 8번째 데이터
0035 00D3                dw   0d300h
0037 0014                dw   01400h
0039 0330                db   3,30h

003B 01          bird_8: db   1          ; 9번째 데이터
003C 00E7                dw   0e700h
003E 0019                dw   01900h
0040 0335                db   3,35h

                 ;-----------------------------------
                 ;
                 ; 데이터 테이블[Rom2]
                 ;
0042 02          bird_9: db   2          ; 10번째 데이터
0043 0080                dw   08000h
0045 0019                dw   01900h
0047 0335                db   3,35h

0049 02          bird_a: db   2          ; 11번째 데이터
004A 0099                dw   09900h
004C 0022                dw   02200h
004E 0232                db   2,32h

0050 02          bird_b: db   2          ; 12번째 데이터
0051 00BB                dw   0bb00h
0053 0024                dw   02400h
0055 0431                db   4,31h

0057 02          bird_c: db   2          ; 13번째 데이터
0058 00DF                dw   0df00h
005A 0021                dw   02100h
005C 0335                db   3,35h

                 ;-----------------------------------
                 ;
                 ; 데이터 테이블[Rom3]
                 ;
005E 03          bird_d: db   3          ; 14번째 데이터
005F 0080                dw   08000h
0061 0024                dw   02400h
0063 0335                db   3,35h

0065 03          bird_e: db   3          ; 15번째 데이터
0066 00A4                dw   0a400h
0068 001D                dw   01d00h
006A 0338                db   3,38h

006C 03          bird_f: db   3          ; 16번째 데이터
006D 00C1                dw   0c100h
006F 0038                dw   03800h
0071 0335                db   3,35h

                 ;-----------------------------------------

0100                     org  0100h

0100 310048      start:  ld   sp,stack
0103 F3                  di              ; 인터럽트 디스에이블
0104 3E82                ld   a,82h      ; &b1 00 0 0 0 1 0
0106 D303                out  (ctrl),a   ; Pa: 출력
0108 AF                  xor  a          ; Pb: 입력
0109 D300                out  (pa),a     ; Pc: 출력
010B D302                out  (pc),a     ;
```

```
                  ;---------------------------------------------

010D              kbd_loop:
010D CD5B01               call kbdscan
0110 30FB                 jr   nc,kbd_loop
0112 E60F                 and  0fh
0114 F5                   push af          ; AF 저장
0115 D300                 out  (pa),a      ; LED on
0117 47                   ld   b,a
0118 210300               ld   hl,bird_0   ; 엔트리 포인트 계산
011B 110700               ld   de,0007h    ; 엔트리 스텝은 7바이트
011E 78           add_pt: ld   a,b
011F FE00                 cp   0
0121 2804                 jr   z,pass
0123 19                   add  hl,de
0124 05                   dec  b
0125 18F7                 jr   add_pt

                  ;---------------------------------------------

0127 F1           pass:   pop  af          ; AF 얻음
0128 47                   ld   b,a         ;엔트리 번호 저장
0129 7E                   ld   a,(hl)      ; 롬 번호
012A 17                   rla              ; 5 비트 시프트
012B 17                   rla
012C 17                   rla
012D 17                   rla
012E 17                   rla
012F E6E0                 and  0e0h        ; 클리핑
0131 B0                   or   b           ; 롬번호 또는 LED 신호
0132 D300                 out  (pa),a

                  ;---------------------------------------------

0134 23                   inc  hl
0135 5E                   ld   e,(hl)
0136 23                   inc  hl
0137 56                   ld   d,(hl)

0138 23                   inc  hl
0139 4E                   ld   c,(hl)
013A 23                   inc  hl
013B 46                   ld   b,(hl)

013C D5                   push de
013D 23                   inc  hl
013E 5E                   ld   e,(hl)      ; E: 반복
013F 23                   inc  hl
0140 56                   ld   d,(hl)      ; D: 타이머
0141 E1                   pop  hl          ; HL: 데이터 스타트 포인터,
                                             BC: 데이터 길이
```

```
                  ;---------------------------------------------

0142 E5           rept:   push hl
0143 C5                   push bc
0144 D5                   push de

0145 7E           speak:  ld   a,(hl)      ; 메모리로 부터 데이터 읽기
0146 D302                 out  (pc),a      ; 포트C로 스피치신호 출력

0148 D5                   push de
0149 15           delay:  dec  d
014A 20FD                 jr   nz,delay
014C D1                   pop  de

014D 23                   inc  hl          ; 메모리 포인터 증가
014E 0B                   dec  bc          ; 길이 카운트 감소
014F 78                   ld   a,b
0150 B1                   or   c
0151 20F2                 jr   nz,speak

0153 D1                   pop  de
0154 C1                   pop  bc
0155 E1                   pop  hl

0156 1D                   dec  e
0157 20E9                 jr   nz,rept

0159 18B2                 jr   kbd_loop    ; 재스타트

                  ;---------------------------------------------

015B 0EFE         kbdscan: ld  c,0feh
015D 210044               ld   hl,keybuf
0160 79           kbloop: ld   a,c
0161 D300                 out  (pa),a
0163 DB01                 in   a,(pb)
0165 47                   ld   b,a
0166 2F                   cpl
0167 A6                   and  (hl)
0168 2010                 jr   nz,kpush
016A 7E                   ld   a,(hl)
016B 2F                   cpl
016C A0                   and  b
016D C49001               call nz,wait
0170 70           kbs2:   ld   (hl),b
0171 23                   inc  hl
0172 79                   ld   a,c
0173 07                   rlca
0174 4F                   ld   c,a
0175 FEFB                 cp   0fbh
0177 20E7                 jr   nz,kbloop
0179 C9                   ret
```

```
            ;--------------------------------------------
017A 70     kpush:  ld   (hl),b
017B 47             ld   b,a
017C CD9001         call wait
017F AF             xor  a
0180 CB09   krloop: rrc  c
0182 3003           jr   nc,kbs3
0184 3C             inc  a
0185 18F9           jr   krloop
0187 07     kbs3:   rlca
0188 07             rlca
0189 07             rlca
018A CB08   kploop: rrc  b
018C D8             ret  c
018D 3C             inc  a
018E 18FA           jr   kploop

            ;--------------------------------------------

0190 AF     wait:   xor  a
0191 E3     waloop: ex   (sp),hl
0192 E3             ex   (sp),hl
0193 3D             dec  a
0194 C8             ret  z
0195 18FA           jr   waloop
            ;--------------------------------------------

0000                end
```

```
---- SYMBOL TABLE ----

ADD_PT   011E          KPUSH    017A
BIRD_0   0003          KRLOOP   0180
BIRD_1   000A          PA       0000
BIRD_2   0011          PASS     0127
BIRD_3   0018          PB       0001
BIRD_4   001F          PC       0002
BIRD_5   0026          REPT     0142
BIRD_6   002D          SPEAK    0145
BIRD_7   0034          STACK    4800
BIRD_8   003B          START    0100
BIRD_9   0042          WAIT     0190
BIRD_A   0049          WALOOP   0191
BIRD_B   0050
BIRD_C   0057          ***** NO ERRORS DETECTED *****
BIRD_D   005E
BIRD_E   0065
BIRD_F   006C
CTRL     0003
DELAY    0149
KBDSCAN  015B
KBD_LOOP 010D
KBLOOP   0160
KBS2     0170
KBS3     0187
KEYBUF   4400
KPLOOP   018A
```

〔프로세스 3〕 크로스 어셈블러를 사용한 컴파일

```
C:\ASM\XASM>
C:\ASM\XASM>xasmz80 bird_com          ←크로스 어셈블러로 컴파일

------------------------------------------------------------
 AVOCET Z80 ASSEMBLER - VERSION 1.04M - SERIAL #00329
 Copyright (C) 1984 by Avocet Systems, Inc.
 All Rights Reserved.
------------------------------------------------------------

***** NO ERRORS DETECTED *****

HIGHEST OBJECT ADDRESS: 0196
USE FACTOR: 000%

ASSEMBLY COMPLETE
```

```
C:\ASM\XASM>dir bird_com.*            ←확인

 Volume in drive C is K. Weon-Jun
 Directory of  C:\ASM\XASM

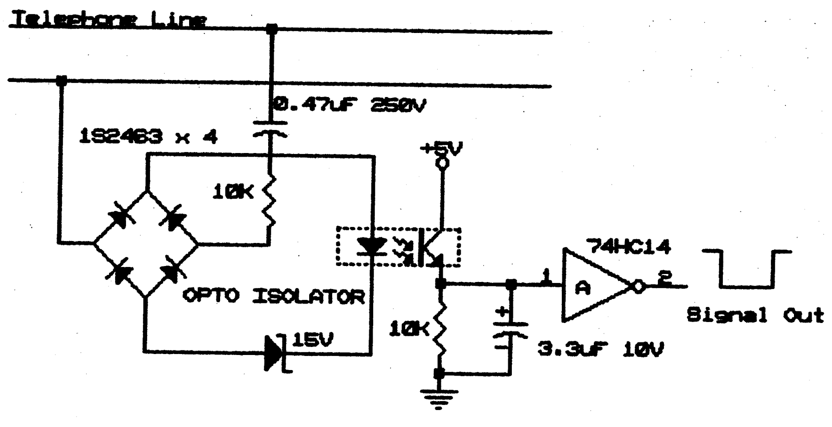
BIRD_COM ASM     6732  12-13-00  12:15a
BIRD_COM HEX      896   3-21-89  11:09a  ←완성
BIRD_COM PRN    13056   3-21-89  11:09a  ←프린터 출력용
        3 File(s)   1245184 bytes free     화일

C:\ASM\XASM>
C:\ASM\XASM>
```

〔덤프 1〕 .HEX 화일의 구성

```
=어드레스=          -코드-       *체크 합*
  ====  --------------------------------**
:10000000C30001000080001C053200009C001F039B
:10001000330000BB002D02340000E8001803420149
:10002000008000180335010090001C03350100B954
:10003000001A03350100D3001403300100E7001952
:100040000335020080001903350200990022023284
:100050000200BB002404310200DF0021033503004D
:10006000800024033503000A4001D03380300C100F1
:030070003803351D
:10010000310048F33E82D303AFD300D302CD5B016D
:1001100030FBE60FF5D300472103001107007BFEFE
:10012000002804190518F7F1477E1717171717E667
:10013000E0B0D300235E2356234E2346D5235E230F
:1001400056E1E5C5D57ED302D51520FDD1230B7828
:10015000B120F2D1C1E11D20E918B20EFE21004408
:1001600079D300DB01472FA620107E2FA0C4900179
:1001700070237907 4FFEFB20E7C97047CD9001AF90
:10018000CB0930033C18F9070707CB08D83C18FA0D
:07019000AFE3E33DC818FADC
:00000001FF
  /종료/
```

〔그림 10〕 전화신호 검출회로




그리고 키입력은 입력과 동시에 동작하고, 키가 눌려진 상태라면 새로 눌러진 키에만 반응하도록 되어 있다. 키패드를 가진 원보드를 제작하는 경우에 응용하면 편리할 것이다.

### (3) 롬에 저장하는 법

음성신호를 확장메모리에 저장하기 위해서는 롬라이터가 필요하다. 롬라이터를 가지고 있는 사람에게는 문제가 되지 않지만 없는 사람은 주위의 가진 사람에게 부탁하기 바란다. 또 롬라이터에 관한 자료는 여러 책에서 얻을 수 있으므로 직접 자작해도 좋을 것이다. 롬에 저장할 때는 필요한 데이터를 빈자리없이 이어서 저장하고 그 시작번지와 길이를 기억하고 있다가 메인 프로그램의 데이터 테이블에 기록하면 된다. 그리고 Z80을 주프로그램을 자작한 롬라이터로 프로그래밍하려는 독자를 위하여 .HEX 화일의 구성을 덤프 1에 제공한다.

### (4) 응용 방법

본기의 응용방법은 무궁무진하다. 키보드 대신 센서를 연결하면 가정의 보안장치로 활용할 수 있다. 발생한 일에 따른 경고음성을 출력할 수 있을 것이다. 또한 다른 프로그램을 개발할 때, 본기에 적당한 메시지를 기록해 두었다가 컴퓨터가 신호를 보내올 때 '디스켓을 넣어 주십시오' 등의 음성이 나오도록 할 수도 있다.

그리고 교육용으로 사용할 경우 이러한 방법도 생각할 수 있다. 희귀한 동물의 울음소리 등을 기록해 두고 재생할 수 있도록 하면 세월이 지나도 음질의 손상없이 그 소리를 들을 수 있다. 그리고 무인응답 장치로도 활용할 수 있다. 특히 전화를 사용하면 많은 응용이 가능하다. 전화를 이용할 경우 가장 어려운 전화신호의 검출회로를 그림 10에 제공한다.

## 마치며

너무 많은 것을 이야기 하려다 보니, 어느 것 하나 충실한 소개가 되지 못한 것 같다. 음성처리분야에 많은 실력있는 분들이 관심을 가지고 참여한다면 멀지않아 정겨운 우리말로 컴퓨터와 대화할 수 있는 날이 다가올 것이라는 기대를 가져 본다.

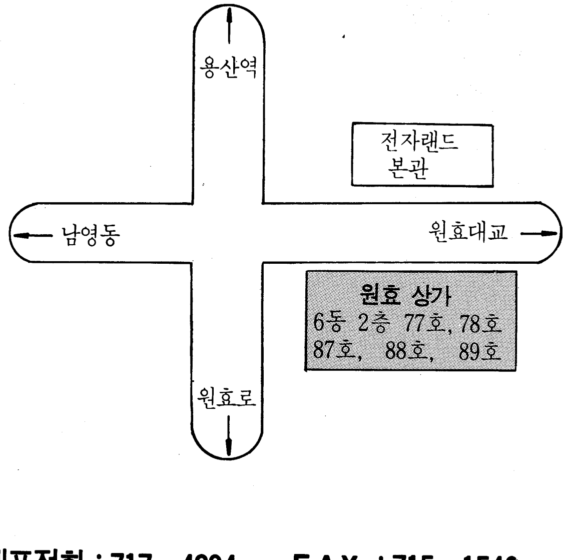
정보사회의 개척자

**하나 컴퓨터**

# 이 전 안 내

귀사 및 귀하의 무궁한 발전을
기원합니다.
당사가 본사를 아래와 같이
이전하였습니다.
많은 지도 편달을 바랍니다.

※ 전화 및 FAX, 사서함은 변경없습니다.




**대표전화 : 717－4894  FAX : 715－1546**
**사 서 함 : 남서울우체국 사서함 47**
**주  소 : 서울시 용산구 원효로 3가**
**원효상가 6동 2층 77, 78, 87, 88, 89호**

# 신제품 안내

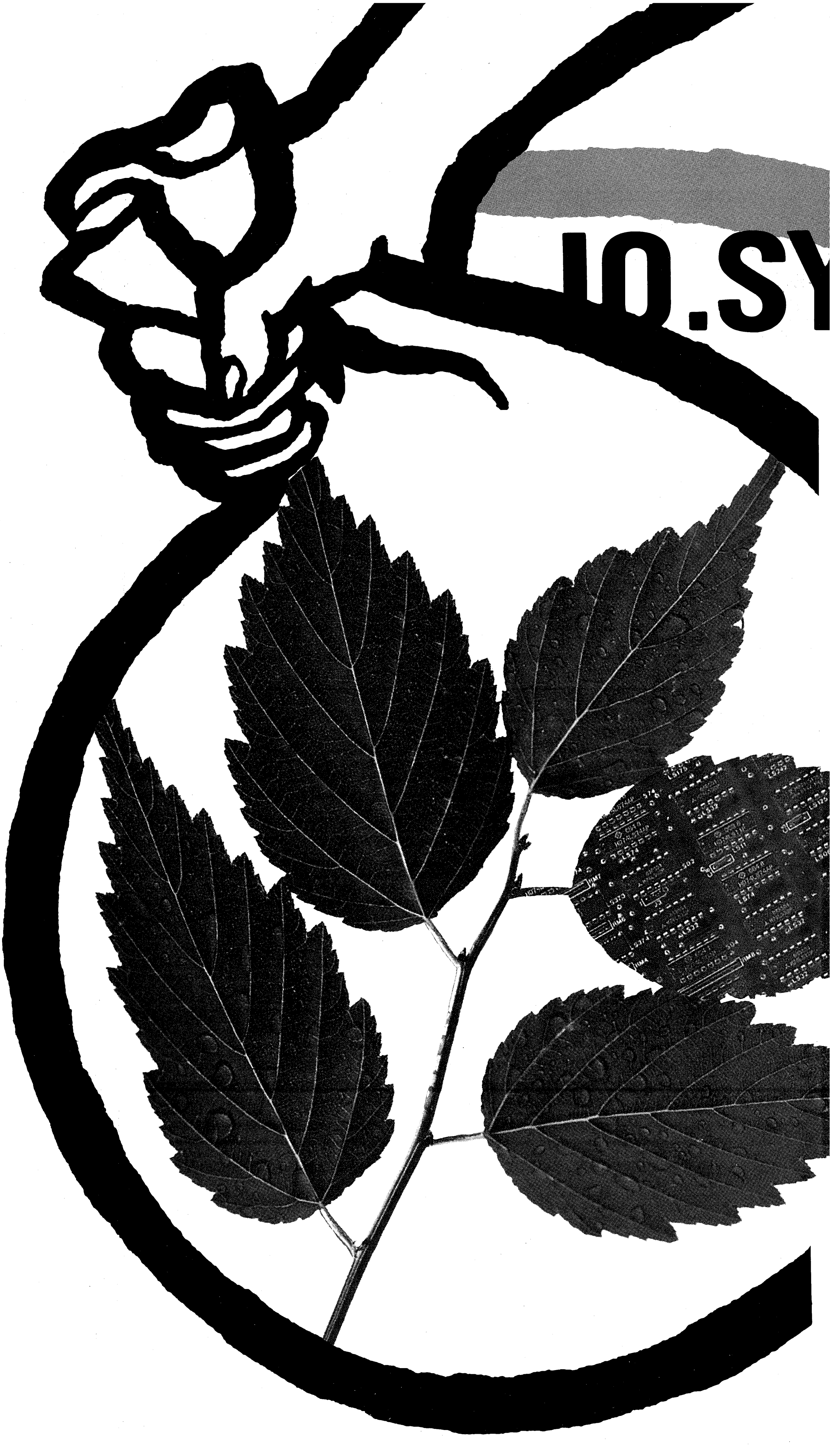
## Top 286 Series

**Model : Top 286 A B C D**

| 속 도 | 12MHz(Zero Wait) |
|---|---|
| HDD | 20MB |
| FDD | 1.2MB |
| Memory | 1MB |
| I/O | 2Serial, 2Parallel<br>1Joy Port |
| 그래픽 | 한글, 한자 |

신제품 개발과 함께
대량생산 체제를 구축하여
수요를 충족시키는 원활한 공급이
가능해졌습니다.
서울지역 및 전국 지역의
대리점 및 딜러를
모집하고 있습니다.

**문의 : 본사 대리점 사업부 (Tel:717－4894)**
**대구 총판 (Tel: (053) 425－2318)**


IO.SY

특집

# S의 분석

**IO.SYS는 시스템 장치를 제어하기 위한 디바이스 드라이버들의 모임이라고 말할 수 있다. 따라서 여기서는 MS-DOS 구조중 가장 중요하다고 할 수 있는 디바이스 드라이버의 기능과 제어방법, 그리고 그 동작의 해설뿐만 아니라 디바이스 드라이버를 작성하는 방법과 이용예에 대해서 알아본다. 지난 3월호에 게재된 'MS-DOS 의 구조'와 5월호 'MSDOS.SYS의 분석'과 더불어 DOS의 구조를 이해하는 데 좋은 자료가 될 것이다.**



**김대순** / 아주대 아미콤

지난 5월호에서 우리는 MS-DOS의 내부구조 중 그 중심이라 할 수 있는 "MSDOS.SYS"의 내부동작에 대해서 알아보았다. 이번 호에는 지난 호의 내용에 이어 "IO.SYS"의 내부구조와 이의 중심적 구조인 디바이스 드라이버의 제어방법 등에 대해서 알아보기로 하자.

"IO.SYS"는 시스템 장치를 제어하기 위한 디바이스 드라이버들의 모임이라 할 수 있는데, 시스템 메이커마다 서로 다른 양식을 취하고 있어, 그 시스템의 장치에 종속적이라 할 수 있다. 따라서 시스템 메이커마다 다른 디바이스 드라이버의 내부동작을 상세히 언급하는 것은 무의미하지만 디바이스 드라이버의 호출법이나 각종 입출력 요구에서 행해야 할 동작은 소프트웨어의 호환성을 고려해서 어느 시스템에서든지 공통적으로 같으므로, 이 같은 부분에 대해서만 중점적으로 알아보기로 하자.

마이크로소프트사의 MS-DOS는 특별한 형태의 컴퓨터 이용자를 위해서 새로운 시스템을 설치해 줄 수 있게 제작되었는데, 이는 새로운 시스템을 부가하기 위해 그 설치에 따른 입출력 사양을 OS의 변경없이 또는 OS의 내부구조를 이해하지 않고서라도 MS-DOS에서 정의하는 간단하고 명료한 데이터 구조의 설계를 통해 새로 설치한 장치를 이용할 수 있다.

이러한 MS-DOS의 특성 때문에 필자는 본고에 또 다른 주제로서 단지 "IO.SYS"의 기능과 그 동작의 해설뿐만 아니라, 사용자의 새로운 장치(마우스나 플로터)의 부가를 위해 디바이스 드라이버를 작성하는 방법과 이용 예를 다루려 한다.

필자가 본고를 통해 소개하는 프로그램중 ADJUSTABLE RAM DISK라는 디바이스 드라이버가 있는데, 이는 필자와 같은 학교에 재학하고 있는 정승민군에 의해 이미 마이크로소프트지에 소개되었던 것으로 독자 여러분의 이해를 돕기 위해 그소스 프로그램을 예로 들어 소개한다. 사실 예로서 MS-DOS V3.2에서의 디스크 디바이스 드라이버를 소개하려 했는데, 그 소스가 너무 크기 때문에 본고의 내용을 이해하고 응용을 목적으로 하는 독자들을 고려해서 이 프로그램을 소개하는 것이다. 이 프로그램은 확장 메모리(EMS : LOTUS / INTEL / MICROSOFT Expanded Memory Specification)를 지원하고 있지만, 본고의 내용과 별다른 관계를 맺고 있지 않다는 생각이 들어 감히 그 부분을 생략했다.

## IO.SYS와 디바이스 드라이버

지난호의 내용중 MS-DOS의 구성에서 가볍게 언급

# IO.SYS의 분석

했듯이 "IO.SYS"는 "MSDOS.SYS"에 의해 조정되는 각종 주변장치를 위한 관리프로그램, 즉 디바이스 드라이버들과 시스템 초기화 작업에 필요한 루틴의 모임인데, 이들은 프린터로의 출력, 보조장치, 콘솔, 디스크, 클럭과의 입출력을 위한 5개의 디바이스 드라이버를 기본적으로 갖추고 있다.

그러나 우리가 사용하고 있는 MS-DOS는 마이크로소프트사가 직접 제작한 것이 아니고 시스템 메이커에 의해서 제작된 것이므로, RS-232C나 리얼타임 클럭을 옵션으로 제공하는 기종에서는 이 부분을 디바이스 드라이버가 옵션으로 제공하고 있다.

이상에서 언급한 것은 기본적으로 제공되는 디바이스 드라이버에 대한 이야기였는데 이 5개의 기본적 디바이스 드라이버는 "IO.SYS"에 포함되어 있어 상주(Resident) 드라이버라 불리운다. 독자 여러분도 잘 아시듯이 MS-DOS버전 2.0 이후에는 새로운 디바이스를 설치하는 것을 지원하기 위해, 또는 기존에 있는 디바이스 드라이버를 새롭게 작성하여 쓸 수 있게 해주는 기능을 가지고 있다. 이는 CONFIG.SYS화일에 "DEVICE=" 명령을 사용하여 새로운 디바이스 드라이버를 설치할 수 있다는 것을 말한다. 그래서 버전2.0 이후의 MS-DOS를 구성가능한 (Configurable) 도스라 부르기도 한다.

본고에서는 새롭게 설치하는 디바이스 드라이버를 "IO.SYS"에 내장되어 있는 상주 드라이버와 구분하기 위해 비상주 드라이버, 또는 설치가능한 드라이버라 부르기로 한다. 앞으로 언급하게 될 디바이스 드라이버의 설명은 일반적으로 시스템 제작업체에서 공급하는 상주 드라이버나 사용자가 작성하거나, 혹은 새로운 주변장치를 제작한 업체가 공급하는 비상주 드라이버에 공통으로 적용된다. 이는 소프트웨어의 호환성과 하드웨어적 장치의 확장을 고려한 MS-DOS 설계자의 기본적 설계가 좋다는 것을 의미하는데, 필자는 여기서 새삼 MS-DOS 설계자의 천재성에 감탄하는 바이다.

## 디바이스 드라이버의 종류

우리는 위에서 "디바이스 드라이버가 IO.SYS에 내장되어 있느냐?"의 여부에 따라 상주 또는 비상주 드라이버로 구분했다. 위의 구분은 단지 MS-DOS의 확장성에 따른 구분일뿐 별다른 의미를 갖지 않지만, 여기서 우리가 이렇게 구분하는 양식은 독자 여러분이 새로운 장치를 시스템에 부가시킬 때 고려해야 할 중요한 것이 되기 때문이다.

우리가 새로 시스템에 부착한 장치는 그 기능에 관계없이 단지 시스템과의 데이터 입출력을 위한 것이라 볼 수 있다. MS-DOS에서도 드라이버가 다루는 디바이스는 그 입출력 데이터 형식에 따라 캐릭터형 디바이스와 블록형 디바이스, 2종류로 구분해서 관리하고 있다.

전자의 케릭터 디바이스는 한번에 1바이트씩 연속되는 문자의 읽기와 쓰기를 수행하는 디바이스를 뜻한다. 이 캐릭터형 디바이스는 화면 1에서 보는 것과 같이 CON, PRN, AUX의 이름으로 불리어지게 되는데, 콘솔(스크린+키보드) 또는 프린터, RS-232C 등이 여기에 해당된다.

그리고 후자의 블록형 디바이스는 시스템에 있는 디스크 드라이버를 뜻한다. 이 디바이스는 블록으로 불리어지는 여러 바이트로 구성된 (보통 물리적 섹터 크기) 형태로 입출력을 수행하는 디바이스를 말한다. 이 블록형 디바이스는 캐릭터형 디바이스와 달리 1개의 디바이스를 1개의 드라이버로 제어하지 않고, 1개의 드라이버로 복수개의 디바이스를 다룰 수 있다. 예를 들면, 우리가 보통 사용하는 2FDD 시스템은 1개의 드라이버로 "A:", "B:" 2개의 디바이스를 다룰 수 있다.

## 디바이스 드라이버의 설치

앞에서 설명했듯이 디바이스 드라이버에는 "IO.SYS"에 처음부터 짜여져 있는 것과 사용자가 나중에 "CONFIG. SYS"에 등록하는 2가지 종류가 있다 했는데, 이들 디바이스 드라이버에는 "MSDOS.SYS"가 이들을 관리하기 위한 디바이스 헤더라는 것이 종류에 관계없이 존재하고 있다. 이 헤더 속에는 각 디바이스 드라이버를 하나의 체인으로 이어주는 링크영역이 존재하고 있어 "IO.SYS"의 시스템 초기화 루틴은 드라이버 작성시 이 링크영역에 다음 디바이스를 가리키는 포인터를 저장하게 된다.

이 포인터는 "IO.SYS"의 시스템 초기화 루틴이 다

〔화면 1〕 디바이스 이름에 의한 디바이스 호출 예

```
A:\>COPY CON PRN
     CON 디바이스로 입력받아
     PRN 디바이스로 출력한 예입니다.
     ^Z
                        1 file(s) copied.
A:\>
```

음 디바이스 드라이버를 로드할 때 필요한 것으로 오프셋과 세그먼트를 포함하고 있다.

"IO.SYS"의 시스템 초기화 루틴은 "MSDOS.SYS"가 메모리로 로드된 후, "CONFIG.SYS"를 해석하여 비상주 드라이버를 설치하고 링크영역을 조정하기 위해 필요한데 이는 화일 단위로 로드되는 드라이버로 화일속의 드라이버 수를 무시하고 그 화일의 첫번째 드라이버의 링크영역만 조정한다.

따라서 우리가 새로운 디바이스 드라이버 작성시 주의할 것은 1개의 화일이 몇개의 디바이스 드라이버로 구성되어 있느냐에 따라 링크영역을 어떻게 정의할 것인가 하는 개념이 달라진다는 사실이다.

화일에 디바이스 드라이버가 하나만 존재할 경우에는 이 링크영역을 −1로 설정하고, 두개 이상일 경우에는 선두에 있는 드라이버로부터 하나씩 다음 드라이버의 오프셋 어드레스를 저장하고, 마지막 드라이버는 −1로 설정하면 된다(**그림 1** 참조).

우리가 드라이버 설치시 주의해야 할 다른 사항은 아래와 같다.

캐릭터형 디바이스 드라이버는 각각 고유의 이름을 갖고 있으며 이용자 코맨드 수준에서 그 디바이스 드라이버를 다루는 것을 화면 1에서 보았다.

그런데 MS-DOS는 캐릭터형 드라이버의 경우, 이미 상주 드라이버에서 사용하고 있는 이름을 비상주 드라이버에서 같은 이름으로 사용하면 그 이름을 사용했던 상주 드라이버는 "MSDOS.SYS"에 의해 무시되는 특성을 가지고 있어, 우리는 이를 이용하여 기존에 제공되어지는 드라이버가 마음에 들지 않을 경우 "IO.SYS"를 수정하지 않고 새로운 디바이스 드라이버로 대체할 수 있다(**그림 2** 참조).

이는 그림 2에서 보듯이 "MSDOS.SYS"는 디바이스 링크로부터 출발하여 제일 먼저 발견되는 디바이스 드라이버를 제어하게 된다는 것을 뜻하며, 이 새로운 캐릭터형 디바이스 드라이버 작성시 뒤에서 설명하게 될 디바이스 헤더의 속성 워드 중 표준입력, 표준출력 디바이스 비트를 1로 설정해야 한다. 이렇게 하지 않으면 CON이라는 이름으로 "CONFIG.SYS"에 의해 등록된 드라이버로 처리되는 것이 아니라 사용자가 무시하고자 했던 CON 드라이버로 처리하게 된다. 이는 매우 불안정한 사용이며, 시스템이 파괴될 가능성을 애써 만들어 주게 되는 결과를 초래한다.

블록형 디바이스 드라이버의 수행은 위의 경우와는 다르다. 즉, MS-DOS는 "CONFIG.SYS"에 의해 등록되는 블록형 디바이스 드라이버를 시스템에 있는 것에 추가하여 판단하게 된다.

따라서 "IO.SYS"에 있는 블록형 디바이스 드라이버가 2대의 디스크 드라이브를 다루고 있다면, 각 디스크 드라이브는 A:와 B:로 구분되어지고, "CONFIG.SYS"에 의해 등록된 새로운 디바이스 드라이버가 만약 3대의 디스크 드라이브를 지원한다면 새로운 추가에 의해 C:, D:, E:로 새로운 드라이브를 구분한다(**그림 3** 참조).

## 디바이스 드라이버의 구조

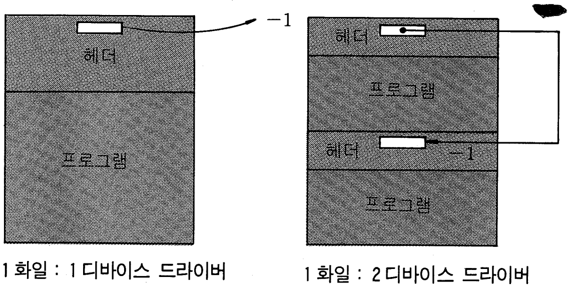
MS-DOS V3.2의 디바이스 드라이버는 멀티태스킹 환경을 지원하기 위해 디바이스 드라이버의 구조를 디바이스 헤더, 스트래티지 루틴(Strategy Routine: 인터럽트 루틴을 수행하기 위한 예비동작)과 인터럽트 루틴(Interrupt Routine: 실제 입출력을 행하는 입출력 명령 처리루틴)의 3부분으로 나누어져 있다(**그림 4** 참조).

왜 멀티태스킹을 위한 구조인지는 앞으로 설명하게 될 다바이스 드라이버의 호출에서 알아보기로 하고, 여기서는 먼저 디바이스 헤더에 대해 살펴보겠다.

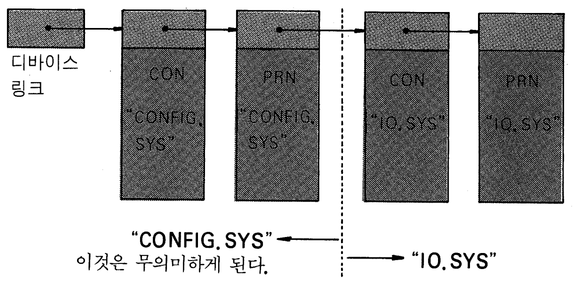
### (1) 디바이스 헤더

디바이스 헤더는 "MSDOS.SYS"가 디바이스 드라이버를 관리하기 위한 정보를 가지고 있는 부분으로 (5월호의 PSP나 EXE형 화일의 헤더처럼) 독자 여러

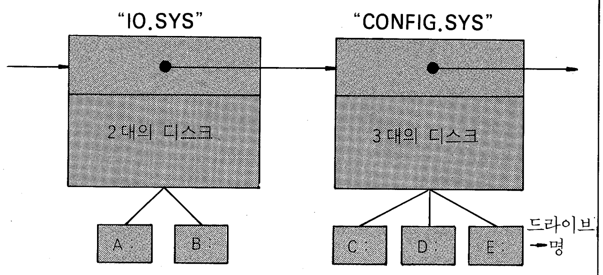
〔그림 1〕 새로운 디바이스 드라이버의 작성 예



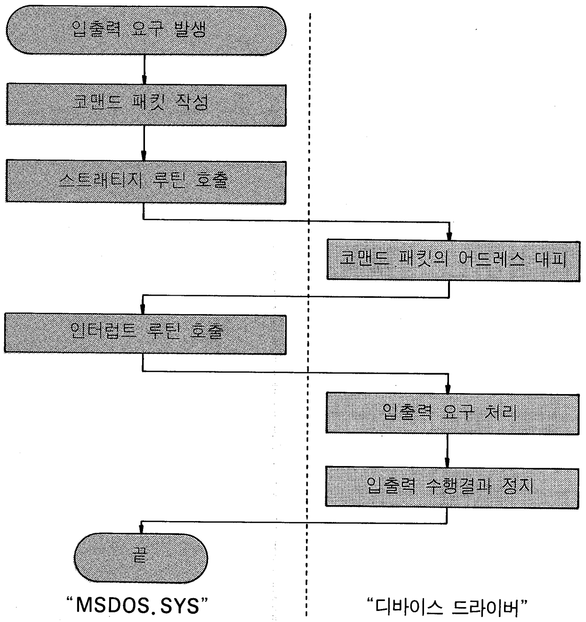

〔그림 2〕 새로운 디바이스 드라이버로의 대체



〔그림 3〕 블록형 디바이스 드라이버의 수행 예




〔그림 4〕 디바이스 드라이버의 동작




분은 이 부분의 상세한 구조와 이것이 담당하고 있는 기능을 이해함으로써 "MSDOS.SYS"가 행하는 동작을 이해할 수 있을 것이다. 이 디바이스 헤더는 **표 1**과 같이 구성되어 있고, 실제 MS-DOS V3.2에서의 소스 프로그램 예는 **화면 2**와 같다.

① 링커영역

표 1과 화면 2의 첫번째 2워드는 다음 디바이스 드라이버의 오프셋과 세그먼트 어드레스를 가지고 있다. 이는 앞의 디바이스 드라이버 설치에서 설명했지만, 부트시 "IO.SYS"에 내장되어 있는 상주 드라이버나 "CONFIG.SYS"에 의해 새로 설치할 비상주 드라이버의 동적인 설치를 위한 것이다. 또, "MSDOS.SYS"가 필요한 디바이스를 선두 디바이스로부터 하나씩 검색하기 위한 것이기도 하다.

② 속성영역

두번째 나타나는 1워드는 9개의 비트로서 그 드라이

〔표 1〕 디바이스 헤더의 구성

| 바이트 오프셋 | 크기 | 내용 |
|---|---|---|
| 00H | 2워드 | 다음 드라이버를 가리키는 포인터 |
| 04H | 1워드 | 디바이스의 속성 |
| 06H | 1워드 | 드라이버 스트래티지 루틴을 가리키는 포인터 |
| 08H | 1워드 | 드라이버 인터럽트 루틴을 가리키는 포인터 |
| 0AH | 8바이트 | 디바이스 이름(C) 또는 유니트 수(B)* |

*(C)는 캐릭터형 (B)는 블록형 디바이스 드라이버를 뜻한다.

〔화면 2〕 디바이스 헤더의 소스 프로그램 예

```
 ;
 ; Device Header for device PRN
 ;

PRNDEV  LABEL   WORD
          DW      TIMDEV,CODE
          DW      CharDev + OutTilBusy + Dev320
          DW      STRATEGY
          DW      PRN0$IN
          DB      'PRN '
```

버가 실행할 디바이스의 유형을 "MSDOS.SYS"가 구별할 수 있게 해주는 구별자 역할을 한다. 이 1워드는 블록형 디바이스와 캐릭터형 디바이스를 구별해 주는 것 이외에도, 각 비트들이 특별한 의미를 가지고 있다. 또, 블록형과 캐릭터형 디바이스를 구분해주는 비트 15에 따라 나머지 각 비트들이 다른 의미를 갖기도 한다.

먼저 캐릭터형 디바이스일 경우와 블록형 디바이스일 경우를 구분해서 정리한 표를 보자(표 2, 3 참조).

**-비트 15**

표에서 보듯이 이 곳의 값은 디바이스의 종류를 나타내는데, 1인 경우 캐릭터형 디바이스, 0인 경우 블록형 디바이스를 뜻한다.

**-비트 14**

이 비트는 디바이스가 IOCTL 시스템 펑션 44H를 통해 디바이스 드라이버를 제어할 수 있는지를 "MS-DOS. SYS"에게 알려주는 구실을 하고 있다.

1인 경우 제어 가능, 0인 경우 제어 불가능을 나타낸다. 여기서 디바이스에 대한 I/O 제어란 디바이스와의 직접적인 입출력이 아니고, 디바이스 드라이버 자체의 상태를 설정하거나 파악하는 것을 의미한다.

**-비트 13**

비트 13은 캐릭터형 디바이스와 블록형 디바이스에

〔표 2〕 캐릭터형 디바이스의 속성

| 비트 | 값 | 의 미 |
|---|---|---|
| 15 | 1 | 캐릭터형 디바이스를 뜻한다. |
| 14 | 1 | 디바이스가 IOCTL(도스 펑션 44H)이 가능한지를 나타낸다. |
| 13 | 1 | 디바이스가 바쁠 때까지 디바이스 호출을 수행한다. |
| 12 | 0 | 예약되어 있다. |
| 11 | 1 | 디바이스가 오픈 / 클로즈를 이해한다. |
| 10-7 | 0 | 예약되어 있다. |
| 6 | 1 | 도스 3.2 기능을 지원한다. |
| 5-4 | 0 | 예약되어 있다. |
| 3 | 1 | 클럭 디바이스이다. |
| 2 | 1 | NUL 디바이스이다. |
| 1 | 1 | 표준출력 디바이스이다. |
| 0 | 1 | 표준입력 디바이스이다. |

〔표 3〕 블록형 디바이스의 속성

| | | 의 미 |
|---|---|---|
| 15 | 0 | 블록형 디바이스를 뜻한다. |
| 14 | 1 | 위의 캐릭터형과 같다. |
| 13 | 1 | FAT ID 바이트를 조사하여 매체를 결정한다. |
| 12 | 0 | 예약되어 있다. |
| 11 | 1 | 위의 캐릭터형과 같다. |
| 10-7 | 0 | 예약되어 있다. |
| 6 | 1 | 위의 캐릭터형과 같다. |
| 5-0 | 0 | 예약되어 있다. |

있어서 그 의미가 달라지는데, 캐릭터형 디바이스에서는 그 디바이스가 바쁠 때까지(Output Until Busy) 디바이스 호출을 수행할 것을 의미하며, 블록형 디바이스에서는 이 비트가 1로 설정되면 그 디바이스가 표준 MS-DOS에서 지원하지 않는 디스크라는 것을 의미한다. 표준 MS-DOS 디스크 사양은 지난 3월호의 MS-DOS의 구조에서 언급했으므로 참고하기 바란다.

**-비트 11**

이 비트는 MS-DOS V3.2에서 새롭게 추가한 IOCTL의 0EH, 0FH, 0CH, 0DH의 기능을 지원하기 위한 것이다.

**-비트 6**

여기는 MS-DOS V2.XX에서 INT 29H(화면에 CTRL-C, CTRL-S를 무시하면서 출력하기 위한 기능)을 사용하기 위한 것이었는데, V3.XX에서는 INT 29H를 지원하지 않으므로 더 이상의 언급은 무의미하다.

**-비트 3~0**

이 비트들은 캐릭터형 디바이스에서만 유효한 것으로 비트 0과 1은 각각 그 디바이스가 표준입력, 표준출력을 의미한다. 또 비트 2와 3은 이 디바이스가 각각 NUL 디바이스와 CLOCK 디바이스라는 것을 나타낸다.

**③ 스트래티지 루틴과 인터럽트 루틴을 가리키는 포인터**

디바이스 헤더중 속성워드 다음의 2개 워드는 각각 그 디바이스 드라이버의 스트래티지 루틴과 인터럽트 루틴을 가리키는 오프셋 어드레스를 가지고 있어, "MSDOS. SYS"가 이를 이용하여 각각의 루틴을 순차 호출하여 입출력 요구를 처리할 수 있게끔 해주는 역할을 담당하고 있다.

**④ 디바이스명**

여기 8바이트의 필드는 캐릭터형 디바이스일 경우, 그 디바이스의 이름으로 사용되고, 블록형일 경우 캐릭터형 디바이스가 8바이트 모두 이름영역으로 사용하는 것과는 달리 선두 1바이트만 유니트 수를 저장하는데 사용하고 있어 나머지 7 바이트를 프로그램 영역으로 사용할 수 있다. 여기서 유니트 수란, 예를 들어 그 드라이버가 3대의 디스크 드라이브를 다룬다면 입출력 요구중 0번 요구, 드라이버의 초기화(INIT)에 의해 돌려진 값을 저장하는 것으로 03H가 된다.

## (2) 스트래티지 루틴

이 스트래티지 루틴은 "MSDOS.SYS"가 디바이스 드라이버를 호출할 때 제일 먼저 호출하는 부분으로, "MSDOS.SYS"는 필요한 기능을 수행하기 위해 디바이스 드라이버를 두번 호출하는데, 그 두번의 호출중 첫번째 호출에 해당한다.

스트래티지 루틴이 하는 기능은 실제로 어떤 입출력 조작도 하지 않으며, 단지 "MSDOS.SYS"의 입출력요구 정보를 가지고 있는 부분의 세그먼트(ES 레지스터)와 오프셋(BX 레지스터)을 디바이스 드라이버의 특정영역에 저장하는 것을 담당하고 있다(화면 3 참조).

이 부분은 사실 현재의 버전(MS-DOS V3.2)에서는 필요가 없는 부분으로 멀티태스킹을 지원하기 위한 MS-DOS 설계자의 배려라 할 수 있는데, 그 이유에 대해서는 뒤의 디바이스 드라이버의 호출에서 알아보기로 하자.

## (3) 인터럽트 루틴

이 루틴은 입출력수행 요구를 실제적으로 처리하는데 필요한 직접적인(하드웨어를 조작하거나, ROM BIOS

〔화면 3〕 MS-DOS V3.2의 스트래티지 루틴

```
;스트래티지는 모든 디폴트 BIO 디바이스 드라이브에 대한
;스트래티지 엔트리 포인트이다.
;이런 모든 것은 변수 PtrSav에 요구되는 헤더에 세이브된
;포인터이다.
;

STRATP  PROC  FAR

STRATEGY:
        mov   word ptr CS:[PTRSAV], BX
        mov   word ptr CS:[PTRSAV+2], ES
        ret

STRATP  ENDP
```

를 호출하는) 코드로 21가지 혹은 그 이하의 프로시듀어들로 구성되어 있고, 이런 일련의 프로시듀어들을 빠져 나오거나 또는 에러를 처리하는 루틴도 포함되어 있다. 이들 각 프로시듀어들의 자세한 설명은 뒤의 입출력 요구 명령 처리루틴에서 자세히 다루겠다.

## 디바이스 드라이버의 호출

위에서 우리는 디바이스 드라이버의 구조에 대해서 비교적 자세하게 알아 보았다. 그러면 이제는 그 디바이스 드라이버가 어떻게 "MSDOS.SYS"에 의해 호출되고, 사용되는지 그리고 위에서 언급을 회피한 사항으로 어째서 그런 구조를 가지게 되었는지에 대해 알아보기로 하자.

"MSDOS.SYS"가 디바이스 드라이버를 호출할 때는 우리가 프로그램 작성시 일반적으로 이용하는 시스템 콜과 같이 특정 레지스터에 명령코드와 파라미터를 주어 호출하는 방식이 아니라, 코맨드 패킷(Command Packet)이라 불리우는 데이터를 구성하여 디바이스 드라이버를 호출하는 방식을 취하고 있다.

즉, 코맨드 패킷을 작성하고 어드레스를 ES:BX에 저장하고 나서 디바이스 드라이버의 스트래티지 루틴을 호출한다. 그러면 이 스트래티지 루틴은 ES 레지스터와 BX 레지스터에 저장되어 있는 코맨드 패킷의 어드레스를 대피시킨 후 제어권을 다시 "MSDOS.SYS"에 전달한다. 이후 "MSDOS.SYS"는 바로 디바이스 드라이버의 인터럽트 루틴을 그 드라이버의 헤더를 이용하여 호출하게 된다. 그러면 "IO.SYS"의 디바이스 드라이버 혹은 새로 설치된 비상주 드라이버는 입출력 요구 정보를 가지고 있는 코맨드 패킷이 이미 디바이스 드라이버가 관리할 수 있는 영역에 존재하므로 코맨드 패킷 중 필요한 코맨드 코드를 읽어 그 명령을 해석 처리하고, 에러 발생시 에러코드를 코맨드 패킷에 정리한다. 그리고 다시 제어를 "MSDOS.SYS"측에 넘겨준다.(그림 4 참조).

예를 들어 "MSDOS.SYS"가 쓰기 요구를 수행하기 위해 블록형 디바이스 드라이버를 호출할 때 수행되는 과정을 알아보자.

7번 이하의 내용은 아직 설명을 구체적으로 하지 못한 부분이므로 독자 여러분께서는 이 과정의 흐름만 이해하면 될 것이다.

① "MSDOS.SYS"는 메모리의 예약된 영역에 코맨드 패킷을 작성한다.
② 그후, 블록형 디바이스 드라이버의 스트래티지 루틴을 호출한다.
③ 블록형 디바이스 드라이버는 ES와 BX 레지스터를 저장하고 리턴한다.
④ "MSDOS.SYS"는 그 드라이버의 인터럽트 루틴을 호출한다.
⑤ 드라이버는 3항에서 저장했던 코맨드 패킷을 가리키는 포인터를 읽어, 이것이 쓰기 요구인가를 결정하기 위해 명령 코드를 읽고 해석한다.
⑥ 드라이버의 디스크 쓰기 루틴은 쓸 디스크 드라이버를 결정하기 위해 유니트 코드를 읽는다.
⑦ 명령이 디스크 쓰기이므로 드라이버는 전송번지, 섹터수, 그리고 코맨드 패킷에서 시작 섹터를 얻는다.
⑧ 드라이버는 첫번째 논리상 섹터번호를 트랙, 헤드, 섹터번호로 전환한다.
⑨ 드라이버는 유니트 코드에 의해서 지정된 드라이브의 시작 섹터에서부터 시작하여 지정된 수의 섹터를 기입하고, 코맨드 패킷에서 지시하는 번지로부터 데이터를 전송한다.
⑩ 전송이 완료된 후, 드라이버는 상태워드내의 종료비트를 1로 설정하여, "MSDOS.SYS"측에 요구의 상태를 보고한다.
⑪ 에러가 발생하면 그 드라이버는 상태 워드의 종료비트와 에러비트를 1로 설정하고, 상태워드 하위 1바이트에 에러코드를 적재한다.
⑫ 마지막으로 드라이버는 "MSDOS.SYS"로 리턴한다.

그러면 어째서 우리가 일반적으로 블록 기능수행을 위해 사용하고 있는 시스템 콜과 같이 입출력 요구를 레지스터에 저장하여 호출하는 방식을 취하지 않고,

복잡하게 따로 코맨드 패킷이라는 것을 작성하여 스트래티지 루틴을 호출한 뒤 부르는가에 대해서 알아보자.

MS-DOS 설계자는 장래에 어떤 필요에 의해서 새로운 기능을 추가하게 될 것을 대비하여 많은 영역을 예약하여 일반 사용자로 하여금 그 영역의 사용을 금지했다.

또한 MS-DOS 설계자는 근간에 선을 보인 멀티태스킹 환경을 기존 OS 수정없이 추가하기 위한 방식으로 설계하였다. 이러한 방식중 장래를 위한 포석(?)중 한가지가 바로 디바이스 드라이버의 "이중호출방식"이다. 멀티태스킹 환경하에서는 입출력이 동시에 일어나는 것을 허용하지 않기 때문에 여러 작업에서 동시 다발적으로 발생하는 입출력 요구를 처리 해줄 수가 없어, 이러한 요구를 하나의 테이블로 만들어 버퍼에 저장하고, 선두 입출력 요구와 링크로 하나의 체인위에 연결한다. 그러면 디바이스 드라이버측에서는 입출력 요구 발생 순으로 이들을 처리해주고, 이 처리를 수행했다는 플래그를 설정하는 것이다. 이러한 방식을 지원하기 위해 스트래티지 루틴을 사용한 이중호출방식이 나온 것이다.

이상의 설명을 잘못 이해하면 마치 현재의 버전(V3.2)이 멀티태스킹을 지원하는 것처럼 보일 수 있으나, MS-DOS V3.2에서는 명백하게 큐의 구조를 갖는 버퍼가 제한되어 있고, 또 각 입출력 요구를 마쳤다는 플래그가 한정되어 있어 불가능하다.

이런 구조는 어디까지나 장래를 위한 포석일뿐 현재로서는 "MSDOS.SYS"의 수행능력을 저하(?)시키는 번잡한 처리과정이라 할 수 있다.

## 코맨드 패킷

위에서 코맨드 패킷이라는 익숙하지 못한 용어로 독자들을 당황하게 했는데, 여기서는 그 구조와 실제적인 예로 코맨드 패킷이 무엇인지 알아보겠다.

코맨드 패킷이란, "MSDOS.SYS"측에서 입출력 요구에 대응하는 정보들로 작성되어 상주 드라이버 혹은 비상주 드라이버에 입출력 요구와 함께 건내주는 파라미터의 모임이다. 이 코맨드 패킷은 선두부터 13바이트의 리퀘스트 헤더라는 것과 각 입출력 요구에 필요한 파라미터 영역으로 구성되어 있다(**표 4** 참조).

**표 5**에서 보듯이 첫번째 바이트는 코맨드 패킷의 길이를 의미하고, 두번째 1바이트를 블록형 디바이스에만 유효한 것으로 블록형 디바이스 드라이버의 유니트중 어느 유니트에 대해 요구되었는지를 나타내기 위한 것이다. 만약 디바이스 드라이버가 3개의 유니트를 지원한다면 이 값은 0, 1, 2가 사용될 수 있다.

〔표 4〕 코맨드 패킷

| | | | |
|---|---|---|---|
| 리퀘스트 헤더 13바이트 | 1바이트 | 레코드의 길이 | → 코맨드 패킷의 길이 (리퀘스트 헤더+파라미터영역) |
| | 1바이트 | 유니트 코드 | → 블록형 디바이스에만 유효하다. |
| | 1바이트 | 커맨드 코드 | |
| | 1워 드 | 상태 | → 입출력작업 결과를 위한 것 |
| | 6바이트 | 예약되어 있다. | |
| | 파라미터용 영역 (각 코맨드 코드에 따라 그 크기가 변한다.) | | |

〔표 5〕 코맨드 코드와 그 기능들

| 코맨드 코드 | 기 능 |
|---|---|
| 0 | 디바이스 드라이버의 초기화(Init) |
| 1 | 매체 검사(Media Check) |
| 2 | BPB(Bios Parameter Block)의 작성 |
| 3 | 읽기 IOCTL(Ioclt Input) |
| 4 | 읽기(Read) |
| 5 | 비파손 읽기(Non Destructive Read No Wait) |
| 6 | 쓰기 상태(Input Status) |
| 7 | 입력 버퍼 클리어(Input Flush) |
| 8 | 쓰기(Write) |
| 9 | 검사와 함께 쓰기(Output With Verify) |
| 10 | 쓰기 상태(Output Status) |
| 11 | 출력 버퍼 클리어(Output Flush) |
| 12 | 쓰기 IOCTL(Output IOCTL) |
| 13 | 디바이스 오픈(Device Open) |
| 14 | 디바이스 클로즈(Device Close) |
| 15 | 제거할 수 있는 매체(Removable Media) |
| 16 | 바쁠 때까지 출력(Output Until Busy) |
| 19 | 일반적인 입출력 제어 요구(Generic IOCTL Request) |
| 23 | 논리 드라이브 사상 요구 설정(Get Drive Map) |
| 24 | 논리 드라이브 사상 요구 얻음(Get Drive Map) |

이 각각의 기능에 대해서는 입출력의 코맨드에서 다루므로 여기서는 생략하자. 그리고, 다음에 나타나는 1워드(2바이트)는 입출력 작업의 결과를 드라이버가 정리하기 위한 것으로 에러나 입출력의 종료, 또는 수행중이라는 것을 나타낸다. 이것을 보통 상태워드라고 부른다(**표 6** 참조).

표 6의 비트 15는 입출력 요구 수행시 에러가 발생한 경우에만 1로 설정되며, 그와 더불어 하위 1바이트는 에러 발행시 에러코드를 적재하게 된다.

또 비트 8은 작업종료 비트로, 1로 설정되면 그 작업

〔표 6〕 리퀘스트 헤더중 상태워드

상태워드

| 15 | 14 | 9 | 8 | 7 |
|---|---|---|---|---|
| ERROR | 예약 됨 | BUSY | DONE | 에러코드(비트 15=1일 때 유효) |

- 0 — 쓰기 프로텍트 되어 있는 디스크에 쓰기
- 1 — 존재하지 않는 유니트 번호
- 2 — 드라이브가 준비되지 않았다.
- 3 — 코맨드 번호가 틀렸다.
- 4 — CRC 에러
- 5 — 코맨드 패킷 길이가 틀리다.
- 6 — 검색 에러
- 7 — 매체 타입이 틀리다.
- 8 — 섹터가 없다.
- 9 — 프린트 용지가 없다.
- A — 쓰기 에러
- B — 읽기 에러
- C — 일반적 에러
- D — 예약되어 있음.
- E — 예약되어 있음.
- F — 디스크 교체가 잘못되었다.

종료 — 코맨드의 실행 종료시 1로 설정.
바쁨 — 코맨드 6, 10, 15에 의해 1또는 0으로 설정.
에러 — 코맨드 실행시 에러가 생길 때 1로 설정.

이 종료되었음을 뜻한다. 그리고 비트 9는 "바쁨(Busy)" 비트로 후에 설명하게 될 상태호출이나 제거가 가능한 매체 호출에만 사용된다.

## BPB

이제 우리가 알아보아야 할 사항은 표 6에서 보는 것과 같은 인터럽트 루틴의 기능들이다. 이들 각 기능에 대해서 알아보기 전에 한가지 이해하고 넘어가야 할 사항은 BPB(Bios Parameter Block)이다.

MS-DOS에서는 다양한 타입의 디스크 위에 화일 시스템을 구축하기 위한 방법으로 **표 7**과 **화면 4**에서 보는 것과 같은 BPB를 작성하여 사용하고 있다.

"MSDOS.SYS"는 디바이스 드라이버가 보내주는 BPB만을 기준으로 해서 이미 지난 3월호에서 언급된 FAT나 디렉토리의 위치를 계산하고, 해당되는 섹터에 액세스 명령으로 디바이스 드라이버의 실행을 요구한다.

그러므로 디바이스 드라이버는 그 디스크의 종류에 대해 전혀 아무런 관여를 하지 않는다. 따라서 블록(보통 섹터) 단위로 입출력이 가능하다면, 즉 랜덤 액세스가 되는 섹터를 가진 구조의 디스크라면 어떤 것이라도 이용할 수가 있다.

〔표 7〕 BPB

| 크 기 | 의 미 | |
|---|---|---|
| 1워 드 | 섹터당 바이트 수 | 섹터의 길이는 바이트로 나타낸다. |
| 1바이트 | 클러스터당 섹터수 | 1 클러스터를 구성하는 섹터의 수 |
| 1워 드 | 예비 섹터수 | 부트용 프로그램이 차지하는 섹터의 수 |
| 1바이트 | FAT의 수 | FAT는 보통 2개를 준비한다. |
| 1워 드 | 루트 디렉토리 엔트리 수 | |
| 1워 드 | 논리상 이미지의 섹터수 | 하나의 디스크에 있는 총 섹터의 수 |
| 1바이트 | 매체 설명 | 화일의 기록형식을 식별하기 위한 것 |
| 1워 드 | FAT당 섹터수 | 하나의 FAT마다 몇개의 섹터를 사용하고 있는가를 나타낸 것 |

〔화면 4〕 BPB의 예

```
BPB             struc

        BytesPerSec     dw      512
        SecPerClu       db      1
        Reserved        dw      1
        nFATs           db      2
        nRootDirEntry   dw      64
        nTotal          dw      9*60
        Media           db      0FCh
        SecPerFAT       dw      2

                ends
```

## 입출력 요구명령 처리루틴

이상에서 우리는 MS-DOS에서의 입출력 요구의 수행절차와 입출력 요구에 필요한 코맨드 패킷, 그리고 실제 입출력 작업을 행하는 디바이스 드라이버의 구조와 그 구조의 근거에 대해서 알아보았다. 여기서는 디바이스 드라이버가 실제의 입출력을 행하는 인터럽트 루틴의 11가지 일반적 기능에 대해서 알아보기로 하자.

디바이스 드라이버로 처리해야 할 기능은 표 6에서 보는 것과 같이 버전 3.20을 기준으로 해서 모두 20개가 있다. 이를 분류하면 대개 11가지의 기능으로 나누어 지는데, 이후의 설명은 이 11가지의 분류에 따른 것이다. 이렇게 분류 기술하는 이유는 각각 요구되는 명

령코드들이 서로 대응하는 기능들로 구성되어 있고, 또 코맨드 코드(명령코드)가 들어 있는 코맨드 패킷의 구조가 서로 같기 때문이다.

이런 11가지의 분류는 아래와 같고, 그와 같은 순서로 코맨드 코드를 포함하고 있는 코맨드 패킷의 형태와 함께 설명하겠다.

① 디바이스 드라이버의 초기화(Init)
② 매체 검사(Media Check)
③ BPB의 작성(Build BPB)
④ 읽기 또는 쓰기(Read / Write)
⑤ 비파손 읽기(Non Destructive Read No Wait)
⑥ 오픈과 클로즈(Open / Close)
⑦ 제거 가능한 매체(Removable Media)
⑧ 상태(Status)
⑨ 입출력 버퍼 클리어(Flush)
⑩ 일반적 IOCTL 요구(Generic IOCTL Request)
① 논리 드라이브 사상 요구를 얻음 / 설정(Get / Set Drive Map)

이들 입출력요구 명령 처리루틴중 어떤 것은 캐릭터형 디바이스에만 또는 블록형 디바이스에만 관계하기도 하고, 몇가지는 두종류 모두에게 필수적으로 있어야 할 것도 있다. 하지만 이런 각각의 처리루틴들이 드라이버의 형에 관계하고 있지만, 이 처리루틴 20개는 모두 실제적인 코드로 존재해야 한다(**리스트** 1 Unimplemented 프로시듀어 참조).

## (1) 디바이스 드라이버의 초기화

코맨드 코드=0

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |
| 1바이트 | 유니트 수 |
| 2워 드 | 끝 번지 |
| 2워 드 | BPB 배열을 가리키는 포인터(B) |
| 1바이트 | 블록 디바이스 번호 |

코맨드 코드 0인 디바이스 드라이버의 초기화 루틴은 디바이스 드라이버가 설치될 때 "MSDOS.SYS"에 의해 단한번 수행하는 것으로 디바이스 드라이버의 초기화를 위한 루틴이다. 비상주 드라이버의 경우 INIT가 호출될 때, "MSDOS.SYS"가 건내주는 코맨드 패킷에는 BPB배열을 가리키는 포인터 영역에 CONFIG.SYS내의 이 디바이스 드라이버가 로드되도록한 줄(DEVICE=XXXX) 에서 등호 다음의 첫번째 문자를 가리키는 포인터가 저장되어 있고, 또 블록형 디바이스 번호 영역에는 INIT가 첫번째 유니트를 정의하기 위한 드라이브 번호가 할당되어 있다.

이 루틴은 드라이버를 초기화한 다음 위의 표에서 보는 것과 같은 3가지의 데이터를"MSDOS.SYS"측에 넘겨준다.

유니트 수는 그 디바이스가 블록형일때는 그 드라이버가 몇대의 드라이브를 지원하는지를 표시한 수가 된다. 이 수는 "MSDOS.SYS"에서 드라이브명으로 할당하기 위한 것이다. 그리고 캐릭터형 디바이스일 경우에는 1로 설정한다.

끝번지는 디바이스 드라이버가 어디서 끝났는지, 즉 메모리상의 어디까지가 그 디바이스 드라이버의 영역인지를 나타낸다. "MSDOS.SYS"는 이 어드레스 이후에 다음의 디바이스 드라이버를 로드한다.

BPB배열을 가리키는 포인터는 각 드라이브에 대한 최초의 BPB상태를 설정하기 위한 것이다. "MSDOS.SYS"는 각 블록형 디바이스 드라이버에서 건낸 BPB를 모아 DPB(Drive Parameter Block : "MSDOS.SYS"는 디바이스 드라이버에서 건낸 BPB를 드라이브의 수만큼 모아 "MSDOS.SYS"용 내부작업 영역을 만든다. 화일 액세스는 DPB를 참고하는 것이다.)를 만든다.

이 포인터는 BPB배열을 가리키는 포인터를 유니트 수 만큼 모아 놓은 배열을 가리키는 포이터이다. 마지막 블록형 디바이스 번호는 "MSDOS.SYS"에서는 아무런 의미가 없고, 디바이스로 보내져 특정 드라이브 유니트에 대해 "MSDOS.SYS"가 현재 사용하고 있는 파라미터를 디바이스가 알 수 있게 한다.

## (2) 매체검사

코맨드 코드=1

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |
| 1바이트 | BPB로부터의 매체 설명 |
| 1바이트 | 되돌려진 값 |

→ 비트 11이 설정되고, 변경된 매체가 보내지는 경우, 보내지는 앞의 볼륨명을 가리키는 포인터

이 매체 검사기능은 블록장치에서만 사용하는 것으로, 오픈, 클로즈, 삭제, 화일명 변경과 같은 화일을 읽거나 쓰기 이외의 드라이브 액세스 호출이 있을 때, "MSDOS. SYS"로부터 호출되어 진다. 이 기능의 목적은 매체가 바뀌었는가를 결정하기 위한 것으로 매체가 바뀌지 않았다고 확인을 할 수 있다면 그 디스크에 관한 FAT를 무효로 처리하지 않는다. 즉 수행효율을 높이기 위한 기능이라 할 수 있다.

"MSDOS.SYS"가 디스크 액세스 호출을 할 경우, 아래와 같은 일련의 과정을 거치게 된다.

"MSDOS.SYS"는 드라이브 문자를 블록형 디바이스의 유니트 번호로 변환시킨다. 그리고, 그 유니트에

서 디스크가 바뀌었는지를 알기 위해 매체검사를 요구하는데, 예전의 매체설명 바이트를 코맨드 패킷에 저장하여 그 디바이스 드라이버를 호출한다. 그러면 드라이버는 다음과 같은 3종류의 값중 하나를 리턴해 주게 된다.

매체가 변경되었음 ······································ 1
매체의 변경여부를 모름 ······························ 0−1
매체가 변경되지 않았음······································ 1

매체가 변경되지 않았을 경우 "MSDOS.SYS"는 디스크 액세스를 계속해서 수행시킨다. 리턴값이 "모른다"이고 디스켓에 수정된 섹터가 존재하나 이 유니트에 대해 아직 디스크로 기입되지 않은 경우에는 "MS-DOS. SYS"는 변경되지 않은 것으로 간주하여 처리한다. 또 변경되었을 경우 그 유니트에 대한 FAT나 디렉토리에 대한 정보는 무효로 하고, (3)에서 설명하게 될 BPB의 작성을 호출하여 수행한다.

BPB가 일단 보내주면, "MSDOS.SYS"는 새로운 BPB 로부터 그 드라이브에 대한 내부구조를 수정하고 디렉토리와 FAT를 읽은 후 액세스를 계속한다.

여기서 한가지 주의해야 할 것은 MS-DOS V3.2 이전의 버전에서 보내진 BPB의 매체설명 바이트가 이전의 매체 설명 바이트와 같은 경우, "MSDOS.SYS"는 디스크 포맷이 같은 것으로 가정하고 내부구조 변경과정을 생략한다. 따라서 모든 BPB는 FAT 이름 바이트에 관계없이 고유의 매체 설명 바이트를 가져야 한다.

## (3) BPB의 작성

코맨드 코드=2

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |
| 1바이트 | BPB로부터의 매체 설명 |
| 2워 드 | 전송 어드레스 |
| 2워 드 | BPB를 가리키는 포인터 |

→ 디바이스 속성 워드의 비트 13의 값에 따라 다용도 영역의 한 섹터, 혹은 FAT의 첫 섹터를 가리킨다.

이 BPB 작성기능은 블록장치에서만 사용되고, 앞에서 언급한 매체 검사기능에서 설명했던 것처럼 BPB 작성기능은 선행된 매체검사 호출이 디스크가 바뀌었거나 또는 바뀌었을 것을 나타낼 때는 언제나 호출될 수 있다. 드라이버는 BPB를 가리키는 포인터를 코맨드 코드 패킷내의 BPB를 가리키는 포인터 영역에 저장해야 한다.

이 포인터는 INIT의 포인터와는 명백하게 다르므로 독자께서는 주의하시길 바란다. BPB 작성기능 호출은 전송번지라는 2워드 포인터를 리턴해주는데, 이 포인터가 가리키는 곳의 내용은 속성워드의 비트 13에 의해 결정된다. 그 비트가 0이면 이곳은 첫번째 FAT의 첫번째 섹터로 이루어지는데, 이곳의 내용을 함부로 변경해서는 안된다. 비트 13은 MS-DOS 표준 디스크 포맷 밖에 다루지 않는 디바이스 드라이버를 의미하는 것이므로, FAT의 위치는 모든 매체에 대해 같아야 한다. 이 비트 13이 1로 설정되는 경우에는 그 포인터는 다용도 영역의 한 섹터를 가리키게 된다. 즉 MS-DOS 표준 디스크 포맷이므로 디스크의 2번째 섹터에 FAT의 최초의 섹터가 존재하게 되고, 그 선두값에 따라 그에 해당하는 BPB를 선택하는 것이다.

## (4) 읽기 또는 쓰기

코맨드 코드=3, 4, 8, 9, 12, 16

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |
| 1바이트 | BPB로부터의 매체 설명 |
| 2워 드 | 전송 어드레스 |
| 1워 드 | 바이트/섹터 수 |
| 에러가 15인 경우 보내진 2워드로 된 이름을 가리키는 포인터 | |

드라이버는 설정된 코맨드 코드에 따라 읽거나 쓰기 호출을 수행하는데(**표 8** 참조), 드라이버가 블록형일 경우에는 섹터단위로 읽거나 쓰기를 수행하고, 캐릭터형일 경우 바이트를 읽거나 쓴다.

여기서 드라이버는 입출력이 완료되면, 상태워드에 에러 여부와 에러코드를 써넣어야 하며, 몇 바이트 혹은 몇 섹터의 데이터가 입출력되었는지를 코맨드 패킷내의 바이트/섹터 수 영역에 저장해야 한다. 이렇게 하는 이유는 도중에 에러가 발생했을 경우를 위한 것인데, 재실행(Retry)을 어떻게 어디부터 해야 하는지 "MSDOS.SYS"에게 알려 주어야 하기 때문이다.

그리고 전송 어드레스는 데이터전송을 시작하는 어드레스를 유지하고 있고, 섹터 시작번호는 전송 어드레스와 마찬가지로 전송을 시작하는 디스크의 논리섹터 번호를 뜻한다. 이는 디스크에서 메모리로 (읽기) 또는 메모리에서 디스크로 (쓰기)의 데이터 전송을 위한 것이다. 덧붙여서 코맨드 코드 9는 **표 9**에서 보는 것과 같이 단지 쓰기기능에 검사하는 기능을 추가한 것이다.

표 8의 코맨드 코드 16은 캐릭터형 디바이스에만 해당하는 것으로 그 디바이스가 "바쁨(혹은 사용중)"을 보낼 때까지 가능한 모든 문자를 출력하도록 하는 것이다. 프로그램에서 이용되고 있는데, 프린터에는 어느 정도의 버퍼가 내장되어 있어 그 버퍼가 넘치지 않는

〔표 8〕

| 코맨드 코드 | 요　　구 |
|---|---|
| 3 | IOCTL 읽기 |
| 4 | 읽기(B/C) |
| 8 | 쓰기(B/C) |
| 9 | 검사와 함께 쓰기 |
| 12 | IOCTL 쓰기 |
| 16 | 바쁠 때까지 출력 |

이상, 즉 바쁘지 않는 상황에서 문자를 계속 프린터로 출력할 수 있게 해준다.

코맨드 코드 3과 12의 IOCTL 읽기, 쓰기란 디바이스 드라이버 자체와의 제어정보를 주고 받기 위한 것인데, 이는 상태워드 중 IOCTL 비트가 1로 설정된 경우에만 호출이 가능하다. 이 IOCTL 기능호출은 에러체크를 수행하지 않는다.

## (5) 비파손 읽기

코맨드 코드=5

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |
| 1 바이트 | 디바이스로부터 읽음 |

이 기능은 캐릭터형 디바이스에만 지원이 되며, 큐구조의 버퍼로부터 하나 앞의 입력문자를 버퍼에서 지우지 않고 읽는 것이다. 즉 하나 앞서 문자를 '보는' 기능이다.

이것은 주로 디스크가 불법적으로 변경되었는지 여부를 결정하는 매체 설명자 바이트에 사용되곤 한다.

## (6) 오픈과 클로즈

코맨드 코드=13, 14

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |

이 기능들은 디바이스 드라이버가 디바이스 헤더 내의 OPEN / CLOSE / RM 속성 비트를 설정하는 경우에 MS-DOS V3.XX에 한하여 호출될 수 있는데, 이것은 현재 화일의 활성 상태에 대해 디바이스 헤더에게 알리도록 구성되어 있다. 블록형 디바이스에서 이 기능은 지역적 버퍼링을 관리할 수 있으며, 디바이스는 오픈된 화일번호의 카운트를 증감시키기 위해 사용될수 있다. 모든 오픈은 디바이스가 오픈 화일번호의 카운트를 증가시키며, 모든 클로즈는 이를 감소시킨다. 이 수가 0일때는 디바이스에 열려있는 화일은 없다는 것을 뜻한다. 블록형 디바이스에서는 FCB 호출을 사용하는 프로그램이 화일을 닫지 않고 열수 있기 때문에 이 오픈 화일번호의 카운트를 못쓰게 되는 경우가 종종 있으므로 사용자는 조심해야 한다. 이런 경우에는 매체가 바뀌고, BPB 작성호출을 할 때 버퍼를 지우지 않고 그 수를 0으로 설정해야 한다.

또한 이 기능은 캐릭터형 디바이스에서 유용하게 쓰이는데, 예를 들어 프린터의 특성(폰트나 페이지 크기)을 IOCTL을 사용하여 초기화하는데 쓰인다. 그리고 사용자는 디바이스에 대해 1회 이상의 오픈을 허용하지 않게 하기 위해 이 기능을 사용하여 오픈 화일번호를 참조할 수 있다.

또한, 모든 프로세서는 STDIN, STDOUT, STDAUX, STDPRN을 액세스하기 때문에 CON, AUX, PRN 디바이스들은 항상 열려 있다. 사용자가 이 디바이스에 대해 클로즈를 행할 경우 어떤 입출력도 불가능하게 되므로 이들에 대해서는 클로즈를 행하면 안된다.

## (7) 제거 가능한 매체

코맨드 코드=15

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |

이 기능은 디바이스 드라이버가 디바이스 헤더내의 OPEN / CLOSE / RM 속성을 활성화 했을 경우에만 블록형 디바이스에 한하여 MS-DOS V3.XX에서 사용이 가능하다.

이 기능 사용후 리턴되는 것은 상태워드의 사용중 비트에 전달되는데, 그 비트가 1이면 매체를 제거할 수 없고, 0이면 제거할 수 있는 매체라는 것을 의미한다.

## (8) 상태

코맨드 코드=6, 10

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |

이 기능은 데이터가 입력이나 출력을 기다리고 있는가를 "MSDOS.SYS"에게 알려주기 위한 것으로 캐릭터형 디바이스에만 사용된다. 코맨드 코드 6의 입력상태 호출은 데이터 입력 버퍼가 비었으면 상태워드의 사용중(Busy) 비트를 1로 설정하고, 비어 있지 않으면 0으로 설정한다. 즉, 버퍼가 비었다는 것은 입력을 기다린다는 것을 의미해 사용중이라는 것을 나타낸다. 코맨드 코드 10의 출력상태 호출은 현재의 출력 디바이스의 상태를 조사하기 위한 것으로 출력 디바이스가 문자를 출력할 수 있는 경우에는 사용중인 비트를 0으

로, 출력이 행해지지 않는 경우 예를 들어 프린터에서 프린터 헤드가 이동할 때 사용중 비트를 1로 설정하여 사용중이라는 것을 알린다.

### (9) 입출력 버퍼 클리어

코맨드 코드=7, 11

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |

이 기능은 캐릭터형 디바이스에만 해당되며 입출력 버퍼의 내용을 취소하는데 사용하고 있다.

### (10) 일반적 IOCTL 요구

코맨드 코드=19

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |
| 1바이트 | 종류 코드 |
| 1바이트 | 기능 코드 |
| 1워드 | [SI]의 내용 |
| 1워드 | [DI]의 내용 |
| 2워드 | 데이터 버퍼를 가리키는 포인터 |

이 기능은 읽기 IOCTL과 쓰기 IOCTL 디바이스 드라이버 기능들을 대체하는데 쓰이고 있다. MS-DOS V3.2에서의 새로운 디바이스 드라이버는 일반적인 IOCTL 을 사용해야 한다.

### (11) 논리 드라이브 사상 요구를 얻음/설정

코맨드 코드=23, 24

| | |
|---|---|
| 13바이트 | 리퀘스트 헤더 |
| 1바이트 | 입력(유니트 코드) |
| 1바이트 | 출력(참고된 마지막 디바이스) |
| 1바이트 | 명령 코드 |
| 1워드 | 상태 |
| 2워드 | 예약됨 |

이 기능은 디바이스 드라이버가 디바이스 헤더내의 워드속성 중 DOS 3.2 비트를 설정하는 경우에만 블록형 디바이스에 한해서 사용이 가능하다. 이 기능은 논리상 디바이스가 현재 사용되는 물리적 디바이스를 소유하고 있는가를 알기 위해 IOCTL의 0EH, 0FH 펑션으로 사용한다.

## 램 디스크

필자가 처음 C언어를 공부할 때는 하드디스크를 보유하고 있지 않아 2대의 플로피디스크 드라이브를 사용하여, A: 드라이브에는 현재 작업하는 프로그램과 컴파일러를 넣고, B: 드라이브에는 헤더 화일을 넣어 사용하곤 했다. 이때 A: 드라이브의 컴파일러는 소스 프로그램을 읽기 위해 A: 드라이브를 액세스하고, 또 프로그램 중 Include 명령으로 헤더 화일을 소스 프로그램중에 포함시키기 위해 B: 드라이브를 액세스하는 방식을 사용하였다.

아직 C언어에 능숙하지 않는 때라 짧은 프로그램도 수십개 이상의 에러를 범하곤 했다. 그리고 에러를 다 고쳤다 생각하고 컴파일 해보면 에러가 쏟아져 나오곤 하여 정말 화나게 하였다. 더우기 필자를 화나게 만든 것은 정성스럽게 고친 프로그램을 컴파일 할 때마다 이 플로피디스크 드라이브라는 녀석은 아주 느긋한 속도로 A:, B:를 번갈아가며 돌아가는 것이다. 그 돌아가는 모습을 멍청한 모습으로 바라보던 필자가 떠올린 생각은 더더욱 멍청하게도 '아! 하드디스크가 있어야 하는데..' 에러는 그렇다고 치더라도 하드디스크가 있어야겠다는 생각을 했다. 그 때 평소 필자를 우스운 녀석이라 생각하고 친절하게 지도 해주던 모 선배께서 필자의 뒤통수를 가볍게 만져 주셨다. '야! 램 디스크를 쓰면 되잖아!'

위의 필자의 경우에서 보듯이 우리는 종종 디스크를 여러번 걸쳐 많은 화일을 읽어야 할 때가 있다. 이때, 화일이 하드나 플로피디스크에 존재하여, 그 액세스 속도가 떨어질때 이들 화일들을 램 디스크에 얹어 놓고 사용하곤 한다. 이 램 디스크란, 메인 메모리의 일부에 만드는 가상적인 디스크를 말하는 것으로, 일반적으로 플로피디스크나 하드디스크보다 그 액세스 속도가 무척 빨라 디스크를 많이 액세스하는 프로그램을 수행시킬 때 많이 이용되고 있다. 이렇게 램 디스크가 속도가 빠른 이유는 독자 여러분도 잘 아시듯이 헤드의 디스크 회전 대기시간 때문이다. 하지만, 자주 쓰이는 화일을 이 램 디스크에 적재하여 프로그램의 수행 효율을 증가 시킨다는 강점을 가진 이 램 디스크도 그 약점을 가지고 있어, 시스템에 전원이 끊길 경우 그 내용이 모두 사라지게 된다. 그러므로 중요한 화일들을 생성할 경우에는 통상 사용되는 플로피디스크나 하드디스크를 사용하는 것이 안전하다.

필자가 여기 소개하는 램 디스크는 이 글의 앞에서도 이야기 했듯이 마이크로소프트지에 이미 소개된 것이지만 그 기능이 다른 램 디스크와 달리 디스크의 크기를 자유자재로 변경하여 사용할 수 있는 기능을 가지고 있어 필자로 하여금 그 소스를 지면으로 공개하게 만들었다. 램 디스크 사용중 이것이 차지하고 있는 메모리가 너무 크거나 작을 경우 그 크기를 변경해야 할 필요가 있다. 이때 기존의 도스에 있는 VDISK.SYS

는 그 크기를 변경시켜 주기위해 CONFIG.SYS의 내용을 고쳐야 하고, 재부트해야 하는 불편이 있다. 여기 이 프로그램은 이런 불편을 해결 해주었다.

프로그램 1은 확장 메모리를 지원해주는 프로시듀어를 생략한 것 이외에는 원래의 소스 프로그램과 거의 같고, 프로그램 2의 EXPAND.C는 원래 DeSmet C v2.4로 작성된 ADJRAM.C에서 램 디스크를 확장하는 부분만을 발췌하여, Turbo C v2.0으로 재구성해 놓은 것이다.

이 프로그램들은 램 디스크가 부트 섹터에서 부트 스트랩용 영역을 사용하지 않는다는 것과 매체 변경이 불가능하다는 것을 이용한다. 램 디스크 드라이버의 읽기 프로시듀어와 쓰기 프로시듀어는 사용하지 않는 부트 스트랩용 영역에 EXPAND.EXE에 의해 생성되는 메모리 블록의 정보(그 메모리 블록의 패러그래프와 크기)를 이용하여 도스에 있는 VDISK.SYS와는 다른 과정의 읽기와 쓰기를 수행한다. 즉 "MSDOS.SYS"에서 건내지는 논리적인 읽기 또는 쓰기의 요구를 실제 블록단위의 읽기와 쓰기로 변경하여 수행하는 것이다(독자께서는 이 부분을 주의하여 보시기 바란다.)

또, EXPAND.EXE는 새로 메모리 블록을 할당할 때마다 매체 설명 바이트를 하나씩 증가시켜 도스가 새로운 BPB를 작성하게 만들고, 그 디스크의 총 섹터수를 변경함으로써 그 크기를 변경시킨다.

〔리스트 1〕 AMDISK.ASM의 소스 리스트

```
; Adjustable Memory Disk Device Driver non-EMS version.
;
; Writed by Ajou Micro COMputer club, S/S Lab. W!
;




include amdisk.inc


cseg  segment para public 'CODE'

driver   proc far

        assume cs:cseg,ds:cseg,es:cseg

; 디바이스 드라이버 헤더의 정의

        dd     -1                       ; 체인에 있어서 마지막 드라이버
        dw     a_block + a_dos          ; 드라이버 상태
        dw     dev_strategy             ; 스트레티지 루틴의 오프셋
        dw     dev_interrupt            ; 인터럽트 루틴의 오프셋
        db     1                        ; 디바이스의 이름 또는 유니트 번호
        db     7 dup ( ? )              ;

bpb_table    dw  bpb                    ; BPB 테이블을 지목

; 스트레티지 루틴에 의한 리퀘스트 헤더의 어드레스 저장 장소

rh_address  dd 1 dup ( ? )

rh_offset   equ word ptr rh_address     ; 리퀘스트 헤더의 오프셋 포인터
rh_segment equ word ptr rh_address + 2  ; 리퀘스트 헤더의 세그먼트 포인터
```

```
; 입출력 요구 처리를 위한 테이블

cmd_table  dw      initialize        ; 드라이버 초기화
           dw      media_check       ; 매체 검사
           dw      build_bpb         ; BPB 작성
           dw      ioctl_read        ; IOCTL 읽기
           dw      read              ; 읽기
           dw      check_input       ; 비파손 읽기
           dw      input_status      ; 입력 상태
           dw      input_flush       ; 입력 버퍼 클리어
           dw      write             ; 쓰기
           dw      write_verify      ; 검사와 함께 쓰기
           dw      output_status     ; 출력 상태
           dw      output_flush      ; 출력 버퍼 클리어
           dw      ioctl_write       ; IOCTL 쓰기

; 디바이스 드라이버 스트레티지 루틴
; ES:BX 로 지정되어 있는 리퀘스트 헤더의 어드레스를
; 프로그램 영역으로 대피 시킨다.
dev_strategy proc

     mov    cs:rh_offset, bx         ; 오프셋의 대피
     mov    cs:rh_segment, es        ; 세그먼트의 대피
     ret

dev_strategy endp

; 디바이스 드라이버 인터럽트 루틴

dev_interrupt proc far

     push    ax                      ; AX 저장
     push    bx                      ; BX 저장
     push    cx                      ; CX 저장
```

# IO.SYS의 분석

```
    push    dx                          ; DX 저장
    push    di                          ; DI 저장
    push    si                          ; SI 저장
    push    ds                          ; DS 저장
    push    es                          ; ES 저장
    cld                                 ; 방향 플래그 클리어
    push    cs                          ; TINY 모드로 전환
    pop     ds                          ; DS:=프로그램 세그먼트
    les     bx, rh_address          ; ES:BX:=리퀘스트헤더의 어드레스
    mov     si, es:rh_command[bx]   ; SI:=리퀘스트 코맨드 코드
    and     si, 0ffh                    ; SI:=코맨드 코드
    add     si, si                  ; SI:=처리테이블에서 코맨드코드로
    call    word ptr cmd_table[si]  ; AX:=실행결과
    lds     bx, cs:rh_address       ; DS:BX:=리퀘스트헤더의 어드레스
    mov     rh_status[bx], ax           ; 상태 워드 정리
    pop     es                          ; ES 복구
    pop     ds                          ; DS 복구
    pop     si                          ; SI 복구
    pop     di                          ; DI 복구
    pop     dx                          ; DX 복구
    pop     cx                          ; CX 복구
    pop     bx                          ; BX 복구
    pop     ax                          ; AX 복구
    ret

dev_interrupt endp

; ---------------------------------------------------------------
; RAM DISK 디바이스 드라이버 입출력 코드들
;
; 모든 레지스터 사용 가능
; ---------------------------------------------------------------

initialize proc                         ; 드라이버 초기화

    lea     ax, word ptr mdisk_data     ; 드라이버 끝(RAM DISK의 시작)
    mov     cl, 4                   ; 을 가리키는 어드레스를 파라그래프
    shr     ax, cl                      ; 단위로 계산한다. AX/=16
    mov     dx, cs                      ; DX := 드라이버 세그먼트
    add     ax, dx                      ; AX := 램 디스크의 세그먼트
    mov     mem_blk_table.par, ax ; 변형을 위해 램디스크 부트섹터로..
    mov     ax, cs                      ;
    mov     es, ax                      ;
    lea     di, word ptr fat1           ; ES:DI:=첫번째 FAT의 어드레스
    mov     cx, sec_per_fat*bytes_per_sec ; CX=그의 크기
    push    di                          ; 첫번째 FAT 어드레스의 저장
    push    es                          ;
    mov     al, cs:bpb_media            ; AL:=매체 설명 바이트
    mov     [di], al                    ; FAT 매체 설명 바이트의 설정
    mov     word ptr 1[di], 0ffffh      ; 나머지는 0FFFFH로 설정한다.
    add     di, 3                       ; 데이터전송,어드레스 3 증가
    sub     cx, 3                       ; 카운터를 3개 감소
    xor     al, al                      ; AL을 클리어한다.
    rep     stosb                       ; FAT의 나머지를 클리어한다.
    pop     ds
    pop     si                          ; DS:SI:=첫번째 FAT 어드레스

    if      fats_per_disk - 1           ; FAT를 2개로 할 경우
      lea di, word ptr fat2             ; 두번째 FAT를 지목
      mov cx, sec_per_fat*bytes_per_sec ; FAT의 크기를 CX에 설정
      rep movsb                         ; 두번째 FAT에 첫번째 FAT의
    endif                               ; 내용을 복사

    mov     ax, cs:bpb_root             ; AX:=디렉토리 엔트리 수
    mov     cl, 5                       ; CL:= AX*2^5
    shl     ax, cl                      ; AX:=디렉토리 영역의 크기
    mov     cx, ax                      ; CX:=디렉토리 영역의 크기
    xor     al, al                      ; AL:=0
    rep     stosb                       ; 디렉토리 영역 클리어
    lds     bx, cs:rh_address           ; DS:BX:=리퀘스트헤더 어드레스
    mov     byte ptr rh_units[bx], 1 ; 리턴 유니트 수
    mov     word ptr rh_end_offset[bx], offset mdisk_data
    mov     rh_end_segment[bx], cs   ; 디스크의 끝번지 계산
    mov     ax, cs:mem_blk_table.siz
    mov     cl, par_per_sec_lg2
    shl     ax, cl
    add     rh_end_segment[bx], ax   ; 드라이버의 끝번지 리턴
    mov     word ptr rh_bpb_offset[bx], offset bpb_table
    mov     rh_bpb_segment[bx], cs   ; 리턴 BPB 테이블 어드레스
    push    cs
    pop     ds
    mov     ah, 19H                     ; 커런트 드라이브를 얻는다.
    int     21H
    mov     dl, al                      ; 그것을 커런트로 설정한다.
    mov     ah, 0EH
    int     21H                         ; DL:=드라이브 번호
    add     al, 'A'                     ; 문자로 변환
    mov     drive, al                   ; 그것을 메세지로..
    mov     dx, offset initok
    mov     ah, 9
    int     21H                         ; 메세지 출력
    mov     ax, s_done                  ; ax:=종료, 에러 없음
    ret

    initok  db 10,13,'Adjustable RAM DISK non-EMS version'
            db 10,13,'Rewrited by Ajou Micro COMputer club'
            db 10,10,13,'initialized as disk '
    drive   db  ' '
            db  ':',10,13,'$'

initialize   endp

; ---------------------------------------------------------------
; 매체 검사
; 램디스크는 일반 플로피디스크와는 달리 매체가 변경되지 않지만, 그
; 매체의 크기는 변할 수 있으므로 우리는 이것을 64K 메모리블록의 수로
```

```
; 검사하기로 한다. 만약 그것이 변했다면 우리는 매체가 바뀌었다고 한다.

media_check  proc                  ; 매체 검사

     lds    bx, cs:rh_address      ; DS:BX:=리퀘스트헤더의 어드레스
     mov    al, cs:bpb_media       ; 매체 설명바이트를 읽어
     cmp    al, rh_media[bx]       ; "MSDOS.SYS"의 것과 비교한다.
     jz     media_same             ; 바뀌었으면 media_same 으로...
     mov    byte ptr rh_check[bx], mc_changed
     mov    ax, s_done
     ret

media_same:
     mov    byte ptr rh_check[bx], mc_same
     mov    ax, s_done             ; AX:=종료, 에러 없음
     ret

media_check  endp

; ---------------------------------------------------------------
;부트 섹터를 읽어 BPB 버퍼에 저장한다.

build_bpb  proc                    ; BPB 작성

     mov    al, cs:bpb_media       ; AL:=매체 설명 바이트
     lds    bx, cs:rh_address      ; DS:BX:=리퀘스트헤더 어드레스
     mov    byte ptr rh_media[bx], al
     mov    word ptr rh_tbl_offset[bx], offset bpb
     mov    word ptr rh_tbl_segment[bx], cs
     mov    ax, s_done             ; AX:=종료, 에러 없음
     ret

build_bpb  endp

; ---------------------------------------------------------------
; 디바이스로 부터 읽기
read   proc                        ; 읽기

     push   es
     pop    ds                     ; DS:=리퀘스트헤더 세그먼트
     mov    di, rh_buf_offset[bx]  ;
     mov    ax, di                 ; 버퍼의 어드레스를 파라그래프
     and    di, 000fh              ; 단위로 표준화한다.
     mov    cl, 4                  ;
     shr    ax, cl                 ;
     add    ax, rh_buf_segment[bx];
     mov    es, ax                 ; ES:DI:=표준화된 버퍼의 어드레스
     mov    dx, rh_buf_size[bx]    ; DX:=읽을 섹터의 갯수
     mov    ax, rh_start[bx]       ; AX:=읽을 섹터중 첫번째 섹터
     mov    bx, -(size mem_blk_table_entry)  ;

find_mem_blk_2_read:               ; 부여받은 논리섹터의 번호가
                                   ;어느 메모리 블록에 있는지
```

```
     add    bx, size mem_blk_table;블록 섹터수로 뺌으로서 구한다.
     sub    ax, cs:mem_blk_table.siz[bx]; 이때 BX는 메모리블록
     jnc    find_mem_blk_2_read ;테이블을 가리키는 포인터가 설정되고,
     add    ax, cs:mem_blk_table.siz[bx];AX는 실제적 시작섹터가 된다.
     push   ax                     ; AX 저장
     mov    cl, par_per_sec_lg2    ; 실제적 시작 섹터를
     shl    ax, cl                 ; 파라그래프로 구한다.

read_set_par:                      ;  AX는 실제적 메모리에 있어서 파
     add    ax, cs:mem_blk_table.par[bx] ;라그래프 어드레스를 갖는다.
     mov    ds, ax
     xor    si, si     ;DS:SI=실제로 읽어야 할 섹터의 어드레스
     mov    cx, cs:mem_blk_table.siz[bx] ;CX:=메모리 블록의 섹터수
     pop    ax                     ; AX:=읽어야 할 섹터수
     sub    cx, ax                 ; CX:=블록에서 읽어야 할 섹터수

read_one_sector:
     or     dx, dx                 ; 모든 섹터가 전송되었는가?
     jz     read_finish            ; Yes, read_finish 로...
     dec    dx                     ; 읽어야 할 총 섹터수 감소
     push   cx                     ; 블록에서 읽어야 할 섹터수 저장
     mov    cx, bytes_per_sec      ; 512 섹터당 바이트
     rep    movsb                  ; 한 섹터를 버퍼로 전송
     pop    cx                     ; CX:=블록에서 읽어야 할 섹터수+1
     mov    ax, ds
     add    ax, par_per_sec        ; 읽어야 할 다음 섹터의 어드레스를
     mov    ds, ax                 ; 파라그래프 단위로 재 설정한다.
     xor    si, si                 ;
     mov    ax, es                 ; DS:SI:=읽어야 할 다음 섹터

     add    ax, par_per_sec        ; 전송해야 할 곳의 어드레스를
     mov    es, ax                 ; 파라그래프 단위로 재설정한다.
     and    di, 000fh              ; ES:DI:=다음 전송할 곳 어드레스
     loop   read_one_sector ;메모리블록의 끝이 아니면 read_one_sector로..
     add    bx, size mem_blk_table_entry

read_set_par_in_loop:
     mov    ax, cs:mem_blk_table.par[bx]
     mov    ds, ax           ; DS:SI:=새로운 메모리블록의 시작 어드레스
     mov    cx, cs:mem_blk_table.siz[bx] ; CX:=메모리블록의 섹터수
     jmp    read_one_sector

read_finish:
     mov    ax, s_done             ; AX:=전송 완료
     ret

read  endp

; ---------------------------------------------------------------
; 디바이스로 쓰기
write      proc                    ; 쓰기
                                   ; 리드 프로시듀어와 같은 알고리즘으로
    push  es                       ; 수행한다. 단지 차이는 버퍼에서
```

```
    pop   ds                  ; 메모리 블록으로 전송방향에 있다.
    mov   dx, rh_buf_size[bx] ;
    mov   si, rh_buf_offset[bx]
    mov   ax, si
    and   si, 000fh
    mov   cl, 4
    shr   ax, cl
    add   ax, rh_buf_segment[bx]
    mov   cx, ax
    mov   ax, rh_start[bx]
    mov   ds, cx
    mov   bx, -(size mem_blk_table_entry)

find_mem_blk_2_write:
    add   bx, size mem_blk_table_entry
    sub   ax, cs:mem_blk_table.siz[bx]
    jnc   find_mem_blk_2_write
    add   ax, cs:mem_blk_table.siz[bx]
    push  ax
    mov   cl, par_per_sec_lg2
    shl   ax, cl

write_set_par:
    add   ax, cs:mem_blk_table.par[bx]
    mov   es, ax
    xor   di, di
    mov   cx, cs:mem_blk_table.siz[bx]
    pop   ax
    sub   cx, ax

write_one_sector:
    or    dx, dx
    jz    write_finish
    dec   dx
    push  cx
    mov   cx, bytes_per_sec
    rep   movsb
    pop   cx
    mov   ax, es
    add   ax, par_per_sec
    mov   es, ax
    xor   di, di
    mov   ax, ds
    add   ax, par_per_sec
    mov   ds, ax
    and   si, 000fh
    loop  write_one_sector
    add   bx, size mem_blk_table_entry

write_set_par_in_loop:
    mov   ax, cs:mem_blk_table.par[bx]
    mov   es, ax
    mov   cx, cs:mem_blk_table.siz[bx]
    jmp   write_one_sector

write_finish:
    mov   ax, s_done
    ret

write     endp

; ---------------------------------------------------------------
; 검사와 함께 쓰기 - 램 디스크는 검사할 필요가 없다.
write_verify  proc

    jmp   write           ; 에러가 없는 것으로 가정한다.

write_verify  endp

; ---------------------------------------------------------------
; 정의하지 않은 기능들을 위한 프로시듀어
unimplemented   proc

    mov   ax, e_command
    ret

unimplemented   endp

ioctl_read      equ     unimplemented
check_input     equ     unimplemented
input_status    equ     unimplemented
input_flush     equ     unimplemented
output_status   equ     unimplemented
output_flush    equ     unimplemented
ioctl_write     equ     unimplemented

; ---------------------------------------------------------------
; 램 디스크 영역
; 섹터 어드레스 계산을 쉽게하기 위해 파라그래프 단위로 정렬한다.

   if (( $ - driver) mod 16 )
      org ( $ - driver) + (16 - (( $- driver) mod 16 ))
   endif

mdisk_data      equ     $           ; 램 디스크의 시작

; 부트 레코드의 초기화

boot_record     db      3 dup(0)    ; 부트 불가능
                db      'AMDISK  '  ; 구분자

bpb:                                ; BIOS Parameter Block
bytes_in_sector  dw     bytes_per_sec; 섹터당 바이트
bpb_sec_per_cl   db     sec_per_cl  ; 클러스터당 바이트
bpb_reserved     dw     1           ; 예약 섹터
bpb_fats         db     fats_per_disk; FAT의 갯수
```

```
bpb_root        dw    128           ; 루트에 있어서 디렉토리 엔트리
bpb_total       dw    min_size_sec  ; 전체 섹터의 갯수
bpb_media       db    1             ; 매체 설명 바이트는...
mem_blk_cnt     equ   bpb_media     ; 메모리블록 카운트를 위해 사용한다.
bpb_fat_size    dw    sec_per_fat   ; FAT 당 섹터
                dw    1             ; 트랙 당 섹터
                dw    1             ; 헤드의 수
                dw    0             ; 감추어진 섹터

mem_blk_table   mem_blk_table_entry <,min_size_sec>
                                    ; 메모리블럭 테이블
end_mem_blk_table equ  $+(size mem_blk_table)*(max_size_sec
                                    /sec_per_blk)
fat1            equ   boot_record + bytes_per_sec
fat2            equ   fat1 + (bytes_per_sec*sec_per_fat)

driver  endp

cseg    ends
        end
```

〔리스트 2〕 EXPAND.C 소스 리스트

```
/*
        Expand Ram Disk non-EMS version

        Writed by Ajou Micro COMputer club.

        This program is coded in Turbo C v2.0
*/


#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include <errno.h>


#define     MAX_SIZE       2043    /* 최대 램 디스크 크기 */
#define     MIN_SIZE       64      /* 최소 크기 */
#define     SEC_PER_BLK    64      /* 32K 블럭씩 증가 */

#define     ENVIR_SEGMENT  0x2C

#define     DOSE_NOERR        0  /* 에러 없음 */
#define     DOSE_INVFUNC      1  /* 틀린 펑션 */
#define     DOSE_ARENA        7  /* 메모리 깨짐 */
#define     DOSE_NORAM        8  /* 메모리 모자람 */
#define     DOSE_INVDRV       15 /* 틀린 드라이브 */

/* 램 디스크에 관한 정의 */

#define   BYTES_PER_SEC   512          /* 섹터당 512 바이트 */
#define   PAR_PER_SEC     (512/16)     /* 섹터당 32 파라그래프 */
#define   SEC_PER_K       (1024/512)   /* K당 1024 바이트 */
#define   DIR_PER_SEC     (512/sizeof(struct DirEntry))
                                       /* 섹터당 디렉토리 엔트리 */

/* 메모리 디스크 부트 레코드 */

struct BootRecord{

    unsigned char jmp[30]; /* 3 바이트만 점프 코드로 사용 */
    struct MemBlockTableEntry {
        unsigned par; /* 메모리블록의 파라그래프 어드레스 */
        unsigned siz; /* 메모리블록 섹터 수 */
    } MemBlockTable[MAX_SIZE/SEC_PER_BLK];
    char   RestOfRecord[BYTES_PER_SEC - 155];

} BootSector;


#define  ident          BootSector.jmp+3
#define  BytesInSector  *((unsigned *)(BootSector.jmp+11))
#define  SecPerCluster  *(BootSector.jmp+13)
#define  BPBReserved    *((unsigned *)(BootSector.jmp+14))
#define  BPBFATs        *(BootSector.jmp+16)
#define  BPBRoot        *((unsigned *)(BootSector.jmp+17))
#define  BPBTotal       *((unsigned *)(BootSector.jmp+19))
#define  BPBMedia       *(BootSector.jmp+21)
#define  BPBFATSize     *((unsigned *)(BootSector.jmp+22))
#define  SecPerTrack    *((unsigned *)(BootSector.jmp+24))
#define  Heads          *((unsigned *)(BootSector.jmp+26))
#define  Hidden         *((unsigned *)(BootSector.jmp+28))
#define  Table_par(i)   *((unsigned *)(BootSector.jmp+30 + i*4))
#define  Table_siz(i)   *((unsigned *)(BootSector.jmp+32 + i*4))


int MemDiskSize,    /* 사용자가 바라는 디스크 크기 ( K ) */
    MemDiskSec,     /* 사용자가 바라는 디스크 크기 ( 섹터 ) */
    MemDiskChg,     /* 현재의 크기와 사용자가 바라는 크기의 차이 */
    FreeSec,        /* 램 디스크에서 사용되지 않는 섹터의 갯수 */
    FirstFATSec,
    LastFATSec,     /* FAT의 위치 */
    FreeCluster;    /* 사용하지 않는 클러스터중 첫번째 클러스터를 지목 */

unsigned char *FAT;
```

# IO.SYS의 분석

```
main(argc, argv)
int argc;
char *argv[];
{
    #define   DRIVENUM    toupper(*argv[1]) - 65
    int j, k;
    struct dfree dfreep[1];

    printf("Expand Memory Disk non-EMS version\n");
    printf("Writed by Ajou Micro COMputer club\n\n");

    /*  이 프로그램이 차지하고 있는 메모리를 해방시킨다. */

    if ( freemem(peek(_psp, ENVIR_SEGMENT)) == -1){
       printf("Error -- %s.\n", sys_errlist[errno]);
     exit(DOSE_ARENA);
}

if ( argc < 2 ) {DisplayUsage(); exit(DOSE_INVFUNC);}

/* 램 디스크로부터 부트 섹터를 읽는다. */

if ( absread(DRIVENUM, 1, 0, &BootSector) == -1 ){
   printf("Error -- %s.\n", sys_errlist[errno]);
   exit(DOSE_INVDRV);
}

/* 그것이 변형가능한 램 디스크인지 검사한다. */

if ( strcmp(ident,"AMDISK  ") != 0 ){
  printf("Error -- that drive is not the adjustable memory
                                 disk!\n");

  exit(DOSE_INVDRV);
}

/* 램 디스크의 크기와 사용가능 크기를 구한다. */

MemDiskSize = BPBTotal/SEC_PER_K;
getdfree( DRIVENUM + 1, dfreep);
if ( dfreep->df_sclus == 0xFFFF ){
  printf("Error -- %s.\n", sys_errlist[errno]);
  exit(DOSE_INVDRV);
}
FreeSec = dfreep->df_total * dfreep->df_sclus;
printf("Drive %c: Total Size: %dK Free Space: %dK\n",
    DRIVENUM + 65, MemDiskSize , FreeSec/SEC_PER_K);

/* FAT를 위한 메모리 확보. 그리고, 램디스크의 FAT를 읽는다. */

FAT = (unsigned char *)calloc(BPBFATSize, BYTES_PER_SEC);
FirstFATSec = BPBReserved;
LastFATSec = FirstFATSec + BPBFATSize - 1;
for (k=0,j=FirstFATSec;j<=LastFATSec;j++,k+=BYTES_PER_SEC)
   absread(DRIVENUM, 1, j, FAT + k);

/* 사용자가 원하는 디스크의 크기를 구한다. */

if ( argc == 3 ){
  MemDiskChg = atoi(argv[2]);
  if ( argv[2][0] == '+' );
  else  MemDiskChg -= BPBTotal/SEC_PER_K;
}
else exit(DOSE_NOERR);

MemDiskSize = MemDiskChg + BPBTotal/SEC_PER_K;
MemDiskSize=((MemDiskSize+(SEC_PER_BLK/SEC_PER_K-1))/
    (SEC_PER_BLK/SEC_PER_K))*(SEC_PER_BLK/SEC_PER_K);
MemDiskSec = MemDiskSize * SEC_PER_K;
MemDiskChg = MemDiskSec - BPBTotal;

printf(" Desired Size: %dK   Desired Change: %dK\n",
                 MemDiskSize, MemDiskChg/SEC_PER_K);

if (MemDiskSize<MIN_SIZE||MemDiskSize>MAX_SIZE) {
  printf("\nMust size of memory disk in K, %d to %d\n",
                              MIN_SIZE,MAX_SIZE);
                    exit(DOSE_INVFUNC);
}

    /* 사용자로 부터 지시받은 대로 램 디스크를 확장 시킨다. */

    if ( MemDiskChg > 0)  Expand();
    else{
       printf ("Size must above 0 !  Nothing to do.\n");
       exit(DOSE_INVFUNC);
    }

    /* 변형된 FAT를 램디스크에 다시 쓴다. */

    if ( abswrite(DRIVENUM, 1, 0, &BootSector) == -1) {
       printf("Error -- %s.\n", sys_errlist[errno]);
       exit(DOSE_INVDRV);
    }
    free(FAT);
    DisplayMemBlockTable();

    /* 램디스크를 확장했으므로 확장한 영역만큼 상주시켜 종료한다. */

    keep(DOSE_NOERR, SEC_PER_BLK * PAR_PER_SEC);
    exit(DOSE_NOERR);
}


/* 새로운 메모리블록을 할당함으로서 램디스크를 확장시키는 펑션 */
```

```
Expand()
{
    int mdisk_dif;
    unsigned segp[1];

    mdisk_dif = MemDiskChg;
    printf("\nAllocating new %dK memory block #%d\n",SEC_PER_BLK/
                                        SEC_PER_K, BPBMedia);

    /* 이 프로그램을 실행시키기 위해, 도스에 의해서 할당된 메모리
       블록의 크기를 변경시킨다.                                   */

    if ( setblock(_psp, SEC_PER_BLK * PAR_PER_SEC) != -1 ){
        printf("Error -- %s.\n", sys_errlist[errno]);
        exit(DOSE_NORAM);
    }
    mdisk_dif -= SEC_PER_BLK;

    /* 매체 설명 바이트의 값을 하나 증가시키고, 새로 확보된 메모리블록의
       파라그래프와 그 크기를 메모리블록 테이블에 정리한다.            */

    Table_par(BPBMedia)  = _psp;
    Table_siz(BPBMedia)  = SEC_PER_BLK;
    FATPut(++BPBMedia, 0, FAT);

    /* 총 섹터의 수를 하나 증가시킨다. */

    BPBTotal += SEC_PER_BLK;

    /*  새로운 메모리 블록을 이제 확보한다. */

    for (;mdisk_dif > 0; mdisk_dif -= SEC_PER_BLK) {
        printf("Allocating new %dK memory block #%d\n",
                    SEC_PER_BLK/SEC_PER_K, BPBMedia);
        if (allocmem(SEC_PER_BLK*PAR_PER_SEC,segp)!=-1){
            printf("Error -- %s.\n", sys_errlist[errno]);
            break;
        }else{
            Table_par(BPBMedia)  = *segp;
            Table_siz(BPBMedia)  = SEC_PER_BLK;
            BPBTotal += SEC_PER_BLK;
            FATPut(++BPBMedia, 0, FAT);
        }
    }
}


FATPut(wd12bits, cluster, FAT)
int wd12bits, cluster;
unsigned char FAT[];
{
    int clloc, clword;

    clloc = 3*cluster/2;
    clword = FAT[clloc] + (FAT[clloc+1] << 8);
    if (cluster & 1) clword=(clword & 0x000f)|(wd12bits<<4);
    else    clword = (clword & 0xf000) | wd12bits;
    FAT[clloc] = clword & 0x00ff;
    FAT[clloc+1] = clword >> 8;
}


DisplayMemBlockTable()
{
     int j;

     printf("\n        Memory Block Table\n");
     printf("Block #   Paragraph    Size(sectors)\n");
     printf("-------   ---------    -------------\n");
     for (j = 0; j < BPBMedia; ++j)
         printf("  %1d %4x %3d\n",j,Table_par(j),Table_siz(j));
}


DisplayUsage(){

printf("Usage: EXPAND drive size\nWhere:\n");
printf("  drive=memory disk drive letter, A to I\n");
printf("  size=size of memory disk in K, %d to %d, in following
                              format:\n", MIN_SIZE,MAX_SIZE);
printf("       +xxx  = increase memory disk size by xxx\n");
printf("       xxx   = set memory disk size to xxx\n");

}
```

〔리스트 3〕 AMDISK.INC의 내용

```
; 디스크의 포맷 지정

min_size_K      equ  64     ; 램 디스크의 최소 크기 64K
max_size_K      equ  2043   ;     "       최대 크기 2043K
sec_per_blk     equ  64     ; 32K 씩 증가
bytes_per_sec   equ  512    ; 섹터당 바이트
sec_per_cl      equ  1      ; 클러스터당 섹터
par_per_sec_lg2 equ  5      ; 섹터당 파라그래프 로그값 32=2^5
fats_per_disk   equ  1      ; FAT의 갯수
sec_per_K       equ  1024/bytes_per_sec   ; K당 섹터
```

```
par_per_sec     equ  bytes_per_sec/16      ; 섹터당 파라그래프
min_size_sec    equ  min_size_K*sec_per_K  ; 램디스크 최소섹터 128
max_size_sec    equ  max_size_K*sec_per_K  ;      "   최대섹터 4086
bytes_per_fat   equ  max_size_sec/sec_per_cl*3/2
sec_per_fat     equ  (bytes_per_fat+bytes_per_sec-1)/bytes_per_sec
K_per_blk       equ  sec_per_blk/sec_per_K

; 리퀘스트 헤더 구조의 정의

rh_length      equ   0              ; 리퀘스트헤더의 길이 영역지정
rh_unit        equ   1 + rh_length  ; 유니트 영역 지정
rh_command     equ   1 + rh_unit    ; 커맨드 코드 영역 지정
rh_status      equ   1 + rh_command ; 상태워드 영역지정
rh_reserved    equ   2 + rh_status  ; 예약 영역 지정
rh_size        equ   8 + rh_reserved ; 리퀘스트 헤더의 크기


; 리퀘스트 헤더의 상태 워드값 정의

s_done          equ   0100h        ; 종료, 에러 없음
s_busy          equ   0200h        ; 수행 중, 에러 없음
e_err           equ   8000h        ; 에러발생
e_protect       equ   e_err+0h     ; 쓰기 금지
e_unknown_unit  equ   e_err+1h     ; 모르는 유니트
e_not_ready     equ   e_err+2h     ; 준비되지 않음
e_command       equ   e_err+3h     ; 모르는 명령
e_crc           equ   e_err+4h     ; 나쁜 CRC
e_bad_length    equ   e_err+5h     ; 틀린 길이
e_seek          equ   e_err+6h     ; 찾지 못함
e_media         equ   e_err+7h     ; 모르는 메디아
e_not_found     equ   e_err+8h     ; 섹터를 찾지 못함
e_paper         equ   e_err+9h     ; 종이 없음
e_write         equ   e_err+ah     ; 쓰기 에러
e_read          equ   e_err+bh     ; 읽기 에러
e_general       equ   e_err+ch    ; 위에서 언급안한 일반적 에러


; 비파손 읽기 기능 호출 파라미터 블록의 정의

rh_read_data   equ     rh_size


; 입출력 파라미터 블록의 정의

rh_media       equ    0 + rh_size        ; 매체 설명 바이트
rh_buf_offset  equ    1 + rh_media       ; 데이터 전송 번지
rh_buf_segment equ    2 + rh_buf_offset  ;
rh_buf_size    equ    2 + rh_buf_segment ; 전송 데이터 크기
rh_start       equ    2 + rh_buf_size    ; 섹터 시작 번호
```

```
m_fized        equ     0f8h              ; 매체:
m_ss9          equ     0fch              ;
m_ds9          equ     0fdh
m_ss8          equ     0feh
m_ds8          equ     0ffh


; BPB 의 작성 기능 호출 파라미터 블록의 정의

rh_bpb          equ    1 + rh_media     ; bpb 의 어드레스
rh_tbl_offset   equ    4 + rh_bpb       ; bpb 테이블의 어드레스
rh_tbl_segment  equ    2 + rh_tbl_offset


; 매체 검사 기능 호출 파라미터 블록의 정의

rh_check        equ    1 + rh_media     ; 매체 검사 결과
mc_changed      equ    -1               ; 매체가 바뀌었다.
mc_maybe        equ    0                ; 모르겠다(바꿨을 것이다).
mc_same         equ    1                ; 바뀌지 않았다.

; 드라이버 초기화 기능 호출 파라미터 블록의 정의

rh_units        equ    0 + rh_size      ; 지원하는 유니트의 수
rh_end_offset   equ    1 + rh_units     ; 드라이버의 끝 번지
rh_end_segment  equ    2 + rh_end_offset
rh_bpb_offset   equ    2 + rh_end_segment ; BPB 배열의 어드레스
rh_bpb_segment  equ    2 + rh_bpb_offset

; 디바이스 드라이버의 상태 정의

a_input        equ     0001h    ; 표준 입력 디바이스
a_output       equ     0002h    ; 표준 출력 디바이스
a_nul          equ     0004h    ; NUL 디바이스
a_clock        equ     0008h    ; 클럭 디바이스
a_dos          equ     0        ; 표준 지원블럭 디바이스(비트13)
a_not_dos      equ     2000h    ; 비표준 블럭디바이스 (비트13)
a_ioctl        equ     4000h    ; IOCTL 기능 제공
a_block        equ     0        ; 블록형 디바이스 (비트 15)
a_character    equ     8000h    ; 캐릭터형 디바이스(비트 15)

; 부트 섹터에서 BPB 다음에 있는 메모리 블록 테이블의 구조 정의

mem_blk_table_entry struc

      par    dw    ?           ; 블록의 파라그래프 어드레스
      siz    dw    ?           ; 메모리 블록의 섹터수

mem_blk_table_entry ends
```

# 전자분야 10년의 역사를 한눈에 확인할 수 있습니다.



**월간 전자과학에서는**
**1979년 1월호부터 1988년 12월호까지**
**총 120권을 장정본으로 묶어 한정판으로 판매합니다.**
**서둘러 구입하십시요.**
**전자분야에 관한 종합적이고**
**체계적인 자료집으로서 지난 10년간의 변천과정과**
**앞으로의 미래를 예측할 수 있게 해드립니다.**

## □ 구입방법

본 장정본의 구입을 원하시는 분은 시중 은행에서
지로용지를 이용하시거나 우체국에서 대체구좌 또는
소액환으로 대금을 송금하시면 됩니다.
은행 지로번호 : 3000667
우체국 대체구좌번호 : 010256-31-0526772

## □ 구입가격

20만원

## □ 문의처

월간 전자과학 영업부 서울시 용산구 갈월동 17
TEL : (02)754-1044 FAX : (02)757-1049

월간 **전자과학** 월간 **퍼스널컴퓨터** 도서출판 **오므사** 관인 **한국전자통신학원**

# 자유롭게 텍스트 화면을 하드카피 해주는 유틸리티 레트릭스

정승민 / 아주대 수학과 HAL

**텍스트 화면을 하드카피 해주는 유틸리티 중 가장 대표적인 것이 Lettrix와 Pizazz이다. 이중 특히 Lettrix는 Pizazz와는 달리 20여개의 다양한 폰트를 가지고 있으며 깨끗하게 화면을 하드카피할 수 있다.**

IBM PC가 PC계에서 강세를 나타내고 있는 커다란 이유중의 하나는 바로 풍부한 소프트웨어에 있다. 업무용을 위시하여, 교육용, 게임, 컴퓨터 랭귀지, 컴파일러, 유틸리티 등등 이루 헤아릴 수 없는 많은 소프트웨어들을 IBM PC는 가지고 있는 것이다. 그런데, 필자는 특히 유틸리티와 통신 소프트웨어에 관심이 많기 때문에 평소에 이런저런 유틸리티를 즐기는 취미(?)가 있다. 그래서 이번호부터 IBM PC의 유저에게 유용한 유틸리티들을 하나씩 소개하고자 한다. 이번 호에서는 프린터에 관계된 유틸리티인 레트릭스(Lettrix)를 소개하겠다.

## 특징

프린터를 소유하고 있는 유저들중에 텍스트화면의 하드카피를 해본 적이 있는 분이라면 아스키코드의 문자들이 제대로 프린팅되지 않고 엉뚱한(?) 문자가 찍히는 것을 경험해 본 적이 있을 것이다.

이러한 현상은 프린터의 코드와 IBM PC에서 사용하는 코드가 다르기 때문에 생기는 것이다. 그러면 이러한 문제점 때문에 텍스트 화면의 하드카피는 정말 어려운 것일까? 그것은 아니다. 이러한 문제점은 소프트웨어로 충분히 해결할 수 있다.

그러한 소프트웨어에는 여러종류가 있는데 대표적인 예로 "Pizazz"를 들 수 있다. 이 소프트웨어는 역상까지 프린팅되는 점이 훌륭하지만 폰트의 부족과 하드카피가 약간 거친(?)면이 있는 것이 옥에 티이다.

이에 비하여 레트릭스는 IBM PC의 모든 아스키 문자를 문제없이 깨끗하게 프린트하여, 화면의 하드카피가 깨끗하고 폰트를 임의로 선택할 수 있다는 점이 그 특징이다. 또 약 20여개의 다양한 폰트를 제공하므로 여러종류의 글씨 형태를 인쇄할 수 있다. 화면 1은 그 대표적인 폰트들을 나타낸다.

그리고 레트릭스는 임의의 폰트를 폰트에디터를 사용하여 자신의 눈에 맞게(?) 고칠 수도 있으며, 램에 상주하는 유틸리티이기 때문에 필요할 때마다 불러내서 사용할 수 있으며, 각종 옵션을 간단한 키조작으로 바꿀 수 있다.

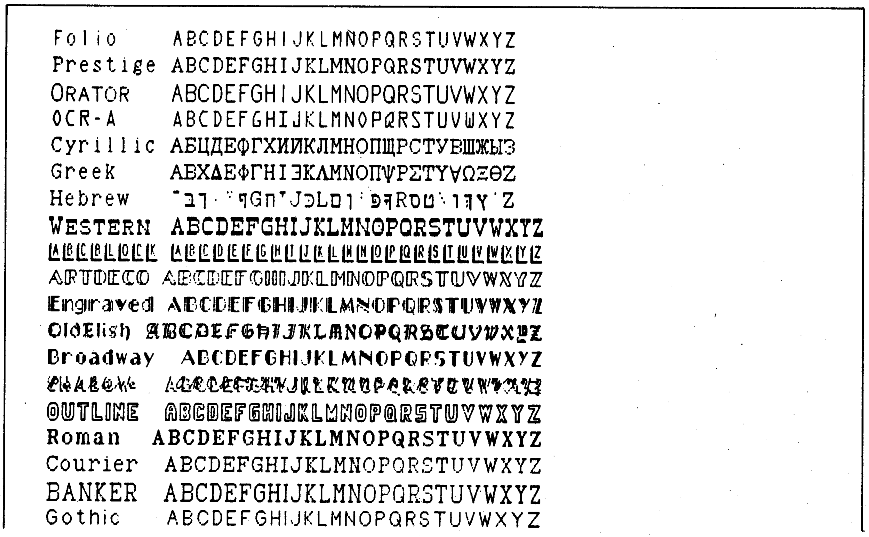
## 레트릭스의 실행

레트릭스 디스켓에는 여러가지 화일이 있는데 다음에 중요한 몇가지 화일을 소개하겠다.

**LX.COM**: 레트릭스 메인 프로그램으로서 램에 상주하는 부분이다.

***.LXH**: IBM Graphics, IBM ProPrinter, Epson FX, LX, JX, RX, Okidata, Panasonic 1092, 1091 등의 프린터들

〔화면 1〕 레트릭스에서 사용가능한 폰트들



의 폰트화일들이다.

*.LXM: Epson MX, Star Gemini 10, 15, Panasonic Sr.Partner, Panasonic 1090 등의 프린터들의 폰트화일들이다.

LXDESIGN.COM: *.LXH, *.LXM의 폰트를 수정하는 폰트에디터이다.

LX-MENU.COM: 레트릭스 옵션을 도스상에서 바꾸어 주는 프로그램으로서 배치화일에서 많이 사용한다.

이제 레트릭스를 실행시키려면 우선 레트릭스 디렉토리를 커런트디렉토리로 한다. 그리고 다음과 같이

```
LETTRIX
```

를 입력하고 리턴한 후에 프린터를 켠다. 그다음 LX 문자뒤에 사용하고 있는 프린터 기종의 코드와 사용할 폰트 이름을 나열한다. 사용할 기종에 관한 코드는 LX를 실행시키면 화면에 나타난다. 그리고 한번에 로드 가능한 최대 폰트수는 7개이다. 예를 들어서 프린터 사용자가 COURIER, PRESTIGE, GOTHIC, GREEK, WESTERN등 5개의 폰트를 로드 하려면 다음과 같이

```
LX COURIER PRESTIGE GOTHIC GREEK WESTERN
```

라고 입력하고 리턴하면 레트릭스는 램에 상주하게 되고 각각의 폰트에는 순서대로 1, 2, 3, 4, 5 등의 번호가 붙는다. 화면 2는 PRESTIGE, SHADOW, ROMAN, COURIER 4개의 폰트를 읽어 들인 후에 인스톨이 끝났다는 메시지가 나온 화면이다.

여기서 한가지 주의할 점은 LX 뒤에 써주어야 하는 폰트화일들이 LX와 같은 디렉토리에 있어야 한다는 점과 프린터가 켜져 있어야 한다는 점이다.

## 메뉴와 키 사용법

자신이 원하는 폰트를 로드하였으면 이제 레트릭스를 활용할 준비는 끝났다. 레트릭스는 램상주 프로그램이기 때문에 핫키 하나면 메뉴를 불러 낼 수 있다. 그러므로 다른 작업 도중에도 실행시킬 수 있다. 핫키는 오른쪽 시프트키를 누른 채로 리턴키를 누르면 화면 3과 같은 레트릭스 메뉴화면이 나타난다. 단 텍스트 모드에서 메뉴를 불러내야 한다. 메뉴 선택을 종료하려면 리턴키를 누르면 메뉴화면이 지워진다.

레트릭스 메뉴가 나온 상태에서 원하는 형태의 프린트 출력을 얻고자 한다면 다음과 같은 옵션키를 누른다. 단, 이때 키는 토글키로서 작동되기 때문에 키를 그냥 누르면 오프되고 시프트키와 함께 누르면 ON이 된다. 그리고 이때는 캡스록이 눌려졌든 안눌려졌든 상관이 없다. 다음은 각 키의 기능을 나타낸 것이다.

P: 프로포셔날 스페이싱. 문자의 폭에 따라서 공백이 정해지는 것이다. 예를 들어서 대문자가 소문자보다 글자폭이 넓고, 'm'자가 'i'자보다 넓다.

J: 오른쪽 줄마추기

N: 컬럼별로 프린팅할 때 ON 시켜놓아야 한다. 예를 들어서 로터스같은 스프레드시트의 화면이나 화일을 카피할 때 필요하다.

D: 두번 찍기(double striking)

B: 볼드체

I: 이탤릭체

H: 2배 높게 프린팅한다.

W: 2배 넓게 프린팅한다.

O: 압축(compressed) 모드로 프린팅한다.

G: 그래픽 문자의 프린팅여부를 결정한다. ON되어 있어야만이 IBM PC의 그래픽문자들이 프린팅된다.

Q: 뒤의 인라인 명령부분에서 나오겠지만 레트릭스에서는 '\'문자가 프린팅의 제어문자로 쓰인다. 특히 화면의 하드카피시에 '\'문자가 화면에 있는 경우(예를 들어서 화일의 경로명(path)에서 자주 나타난다) 하드카피가 엉뚱하게 되기 때문에 이 문자 '\'를 일반문자로 인식시키는 옵션이 바로 이 'Q'옵션이다. 즉 '\'문자가 있는 화면의 하드카피시에는 이 옵션을 오프시켜야 한다.

'1'-'7': 레트릭스는 한번에 최대 7개의 폰트를 램에 상주시킬 수 있는데 램에 상주시킨 폰트들을 숫자키로 전환하여 키대신 사용할 수 있다. 예를

〔화면 2〕 4개의 폰트를 로드한 후의 인스톨 화면

```
A:\>lx prestige shadow roman courier
Reading prestige shadow roman courier
LETTRIX Letter Quality Software              (C) 1985 Hammerlab Corporation

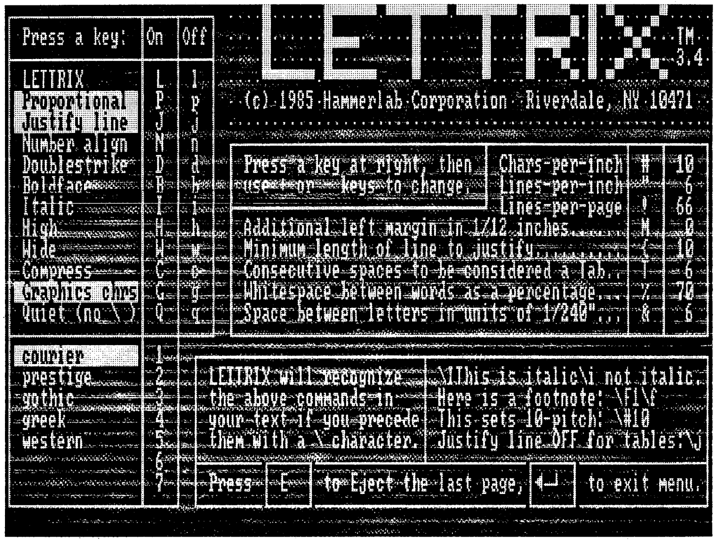
Duplication for simultaneous use by others violates U.S. copyright statutes.

Now, use your wordprocessing or other software and print as you would normally!

While depressing the right-hand shift key, press [↵] to see the LETTRIX menu.

A:\>
```

〔화면 3〕 레트릭스 메인메뉴




들어서 5개의 폰트를 읽어 들이면 앞에서부터 차례로 1, 2, 3… 하고 폰트에 번호가 붙는다. 처음에는 1번 폰트(앞의 예에서는 COURIER)가 지정되어 있지만 레트릭스 사용도중에 WESTERN 폰트를 사용하고 싶으면 메뉴 모드에서 5번키를 누르면 WESTERN 폰트가 지정되는 것이다. 폰트의 번호는 레트릭스 메뉴 화면 왼쪽 밑부분에 나와 있다(**화면 3** 참조).

L: 레트릭스 기능을 사용하느냐 안하느냐의 선택을 나타낸다.

프린트시 1인치당 글자나 행수, 1페이지당 행수 등의 수치를 주어야 하는 것들이 있는데 이러한 것들의 지정은 직접 수치를 입력하는 것이 아니라, 해당 옵션키를 누른 후에 오른쪽 숫자키 패드상의 '+' 회색키로서 수치를 1씩 증가시키고 '-' 회색키로서 1씩 감소시키게 된다. 이러한 방법으로 사용되는 옵션은 다음과 같다.

#: 1인치당 글자수(CPI, Characters Per Inch)
": 1인치당 행수(LPI, Lines Per Inch)
!: 1페이지당 행수(LPP, Lines Per Page)
M: 왼쪽에 공백을 만들어 준다. 단위는 1/12인치가 기준이 된다. 예를 들어서 6을 지정하게 되면 6/12인치 즉, 1/2인치만큼 왼쪽에 공백을 만들어 준다.
{: 줄맞추기(Justify)할 때의 최소 행길이이다. 스트레칭(줄맞추기할 때에 단어사이에 공백을 넣음으로써 행길이가 길어지는 것)을 방지하기 위해서는 작게 잡는 것이 좋다.
T: 탭으로 여기는 연속공백이다. 이 수치보다 작은 공백이 있으면 이 공백은 무시되어 없어진다.
%: 단어사이의 공백을 지정한다.
&: 글자사이의 공백을 나타낸다.

이렇게 하여 원하는 메뉴를 선택하여 출력하고자 할 때는 메뉴상태를 빠져 나온다. 만약 DOOLY.TXT라는 화일을 선택하여 다음과 같이

```
COPY DOOLY.TXT PRN
```

라고 하면 원하는 형태의 프린터 출력이 나온다.

## 인라인 명령

이 명령은 레트릭스의 프린팅 옵션을 메뉴상태에서 선택하도록 하지 않고 텍스트화일 내에서 조정할 수 있도록 하는 명령어이다. 그러므로 자신이 프린트하기 원하는 텍스트 화일자체에서 사용하면 된다. 이 명령은 '\'문자뒤에 옵션 문자를 주면 된다. 첫번째의 토글옵션은 대문자를 '\'뒤에 쓰면 ON이고 소문자를 주면 OFF가 된다. 그리고 '1'에서 '7'까지의 숫자를 주면 폰트를 선택하게 된다. 그 외 인라인 명령에서 추가지원되는 옵션으로 밑줄문자(Underline), 윗첨자(Superscript), 아랫첨자(Subscript) 등이 있다.

그 외 수치가 필요한 옵션은 수를 써 주면 되는데 '{'이 인라인의 명령에서는 '}'로 되었고 나머지는 같다.

**화면 4**는 레트릭스 OFF상태에서 프린트한 예이고 **화면 5**는 레트릭스 ON 상태에서 프린트한 예이다. 예를 통하여 레트릭스가 어느 정도의 글씨체를 제공하는지 비교가 될 수 있을 것이다.

## 폰트 디자인

인자체에 대한 욕심이 많은 독자가 있다면 자신의 폰트를 만들어 사용하고 싶어질 것이다. 이러한 극성(?)맞은 사용자를 위해 레트릭스에서는 사용자가 직접 자신의 폰트를 디자인할 수 있게는 하는 기능으로 LXDESIGN을 제공하고 있다. 자신만의 폰트를 만들고 싶다면 기존의 폰트를 다른 이름의 화일로 복사한 후에 그 화일을 수정하면 된다. 예를 들어서 PRESTIGE를 MYFONT라는 화일이름으로 복사한 후에 자신이 직접 폰트를 에디트하려면 다음과 같이

```
COPY PRESTIGE.LXM MYFONT.LXM
LXDESIGN MYFONT.LXM
```

라고 하면 된다.

이렇게 하면 **화면 6**과 같은 폰트에디터의 상태로 들어가게 된다. 그 후는 에디터내의 좌측의 Help화면을 참고하면서 작성하면 된다. 임의의 키를 고르려면 리턴키를 누른 후에 해당 키를

〔화면 4〕 레트릭스 오프상태

```
\"5
\1
\W Lettrix \w is very \Bpowerful\b printing utility
introduced by \H Darth Vader. \h

\2 It can change Font like this case. \1

 You can \Uunderline\u anywhere!

 And change italic like \Ithis.\i

 Any punctuation mark after; a ^ will print as an
accent:
             Espan^~ol

 You can s-t-r-e-t-c-h a line by ending it with a
backslash followed by dashes:\-------------------

 To align columns, turn off justification with j:
\j
             Apples        69c      1/2 lb.
             Peach        519c      3/2 lb.

To print text with a ragged-right margin, turn
off LETTRIX justification as well as your
software's own justification.
\J
```

〔화면 5〕 레트릭스 온상태

**Lettrix** is very **powerful** printing utility introduced by Darth Vader.

It can change Font like this case.

You can <u>underline</u> anywhere!

And change italic like *this.*

Any punctuation mark after; a ^ will print as an accent:

Español

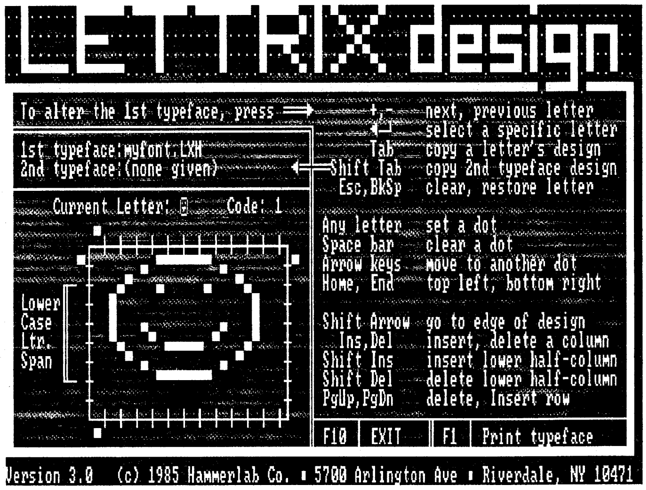
You can s-t-r-e-t-c-h a line by ending it with a backslash followed by dashes:

To align columns, turn off justification with j:

| | | |
|---|---|---|
| Apples | 69c | 1/2 lb. |
| Peach | 519c | 3/2 lb. |

To print text with a ragged-right margin, turn off LETTRIX justification as well as your software's own justification.

〔화면 6〕 폰트에디터 LXDESIGN




누르면 된다. 키보드에 없는 문자를 에디트하려면 〈ALT〉키를 누른 채로 숫자판의 키로써 아스키 코드값을 입력해 주면 된다. 그리고 에디트를 마치면 〈F10〉키를 눌러 고친 내용을 저장시킨다.

## 배치화일에서

레트릭스가 램에 있는 상태에서 오른쪽 ·시프트키와 리턴키를 눌러서 메뉴상에서 레트릭스 옵션을 바꾸는 법을 앞에서 알았다. 이제는 도스상에서 (주로 배치화일상에서) 옵션을 바꾸어 보자. 이렇게 하기 위해서는 LXMENU.COM이 필요하다. 예를 들어서 'J'옵션을 주고 싶으면 도스상에서 다음과 같이 하면 된다.

```
LX-MENU J
```

이번 호에서는 레트릭스라는 프린팅 유틸리티를 알아보았다. 부디 많은 유저들이 이 유틸리티를 활용한다면 좋겠다.

그리고 필자에게 질문사항이 있는 유저는 필자가 운영(?)하고 있는 BBS로 콜을 해서 메일을 남기시기 바란다. 전화번호는 (02)797-0825이고 밤 9시부터 아침 7시사이에 콜 하면 된다.

# 전화번호부 프로그램(Ⅱ)

박성현/ 디베이스 프로그래머

이번호에서는 지난호에 이어 전화번호부 프로그램의 READ.PRG를 소개하고자 한다. 이번 프로그램은 데이터화일을 읽어 들여 처리한 후 성명이나 회사순의 원하는 형식대로 전화번호부를 열람하거나 전화를 걸어주거나 또는 수정, 삭제와 열람을 중지하도록 한다. 그러면 이를 행번호에 따라 분석해 보자.

## READ.PRG의 분석

**(7-10) SET FUNCTION 9 TO [−1;]**

F9 키로 '–', '1', 〈Enter〉키를 차례로 입력하는 것처럼 대체시켜 주게 한다. 이 FUNCTION 정의명령은 F2에서 F10까지 가능하며 F1은 디베이스 자체의 HELP 기능에 사용하므로 정의할 수 없게 되어 있다. 여기서는 4가지만 정의하였으나 독자의 필요에 따라 모두 정의해서 사용하면 프로그램을 더 편리하게 실행시킬 수 있을 것이다. 모든 문자를 쓸 수 있으나 〈Enter〉는 표기할 수 없으므로 ';'를 사용한다.

**(11) SET COLOR TO W/N+, W/N+**

GET에 의해 반전되는 색도 검은 배경에 밝은 흰글씨로 나오게 하기위해 W/N+를 사용한다.

**(12) CLEAR**

열람화면이 나오기전 화면을 깨끗이 지운다.

**(13-36) TEXT...ENDTEXT**

디베이스에서는 TEXT 다음의 라인부터 ENDTEXT 바로 전까지의 라인의 내용은 화면에 곧바로 디스플레이해주므로 여기서는 SAY에 의해 라인을 그리지 않고 프로그램내에서 직접 라인을 그려서 직접 그린 내용이 디스플레이되게 했다. 앞의 명령 (12)행에서 CLEAR라는 명령을 사용하여 현재의 커서위치를 0,0로 만들었기 때문에 TEXT 다음의 첫번째 라인은 1,0에서 시작하게 된다. 만약 @ 3,79라는 명령 다음 TEXT라는 명령이 나오면 TEXT 다음줄(디스플레이 되기 시작하는 부분)은 4,0부터 시작하여 디스플레이되기 시작한다.

**(38-39) ndx1=[성명]...ndx2=[회사]**

변수의 내용을 정의한다. 변수의 변화에 따른 변수를 선택하기 위해 같은 변수명에 숫자의 연속을 주었다.

**(42) ord=[?]**

다음의 LOOP 안으로 들어가게 하기위해 1과 2를 제외한 아무 내용으로 변수를 정의했다.

**(43-47) DO WHILE...ENDDO**

변수 ord의 내용 입력시 잘못된 입력을 다시 입력받기 위한 루프이다.

**(43) DO WHILE .NOT. ord$[12]**

ord$[12]는 ord=[1] .OR. ord=[2] 라고 해도 상관없다. 여기서 '$'의 역할은 aaa$bbb라고 했을 때 aaa라는 변수의 내용이 bbb라는 변수에 존재한다면 참값이 되고, aaa라는 변수의 내용이 bbb에 포함되어 있지 않으면 거짓이 되게 한다. 그러므로 ord라는 변수가 [1] 또는 [2]가 아니면 ord$[12]은 거짓이 되지만 앞의 .NOT.에 의해 다시 참이 된다. 그러므로 ord 변수가 [1] 또는 [2]가 아닌동안 계속 된다.

**(44) ord=[?]**

잘못 입력했을 경우에 계속해서 '?'가 입력받을 위치에 보이게 하기 위해 변수를 재지정한다.

**(45-46)**

ord 변수를 새로 입력받는다. PICTURE [9]는 입력을 받되 숫자만 받고자 할 때 [9]를 사용한다. READ는 지금까지 나온 GET을 입력할 때까지 기다리고 그 내용을 받아들인다.

**(49) SET ORDER TO 〈n〉**

열려있는 인덱스화일 중 n번째 화일을 마스터 인덱스화일로 지정한다. n이 2라고 하면 위의 인덱스화일 중 CCOMP. ndx가 마스터 인덱스화일이 되어 그순간부터 CARD.dbf의 배열은 회사순으로 바뀌게 된다.

다른 배열의 인덱스화일로 바꾸고자 할 때 이와같이 인덱스화일을 그때 새로 열지 않고 한꺼번에 같이 열어 놓은 후 마스터 인덱스화일만 바꾸어 주면 화일을 열고 닫는 빈번한 시간을 줄이게 되며, 또 항상 같이 열려 있어야 데이터가 바뀌거나 추가될 때 인덱스화일도 함께 재배열되게 된다.

&ord는 ord라는 변수에 들어 있는 내용을 명령으로 인식하게 해주는 기능을 갖게 해준다. &(macro)의 기능은 변수의 내용이 문자인 경우에 기억변수의 내용을 치환하는데 사용한다. 다음의 예를 실행 해보면 이해가 빠를 것이다.

```
. aaa=[번호]
. bbb=[전화 &aaa.부]
. ? bbb
```

```
전화 번호부
. ccc=[CLEAR]
. &ccc
```

위에서 &aaa 다음의 '.'은 공백이 없는 문자열에 기억변수를 추가할 때 메모리 변수의 끝임을 표시하는 것이다. &ccc 명령어도 치환하여 사용할 수 있다.

**(50) ans = SPACE(20)**

성명이나 회사의 입력을 위해 변수를 미리 지정해 놓는다.

**(51) ndx&ord.**

@ 11, 21 SAY [열람하고자 하는] +ndx&ord.+[를 입력 하시오.] 위에서 ord의 변수값이 '1'로 정해졌다면 ndx1의 변수값인 [성명]이 디스플레이 될 것이며 '2'로 정해졌다면 ndx2의 변수값인 [회사]가 디스플레이 된다. 또다른 방법으로 '@ 11, 21 SAY [열람 하고자 하는]+IIF(ord=[1], [성명], [회사])+[를 입력 하시오]'가 있다.

어떤 방법이 더 효율적인가 생각해 볼 때 &(macro) 함수의 경우는 상당히 시간을 소모하는 명령이므로 가급적 사용을 피하는 것이 좋다. 여기서는 매크로 함수를 설명하기 위해 사용했을 뿐 실제 프로그램에서는 위와 같이 IIF()를 사용하면 더 빠르고 간단한 프로그램을 작성할 수 있다. 만약 ord 변수의 변화가 여기서 처럼 '1'이나 '2'가 아니고 1-10 정도라면 여러개의 IIF()를 사용하는 것보다 &(macro) 함수를 사용하는 것이 훨씬 효율적일 것이다.

**(52-53) @ 14, 26 GET ans...READ**

찾고자 하는 성명 또는 회사의 입력을 받는다.

**(54) ans = LTRIM(RTRIM(ans))**

변수 ans의 값 왼쪽, 오른쪽에 있는 공백을 제거시킨다. LTRIM()은 왼쪽의 공백을 제거하고, RTRIM()은 오른쪽의 공백을 제거한다. 여기서 공백을 제거하는 이유는 데이터를 찾을 때 다음의 결과가 생기기 때문이다.

```
? [홍 길동] = [홍    ]
.F.
? [홍 길동] = [홍]
.T.
```

첫번째 비교에서는 당연히 거짓값이 나오지만 두번째 비교에서는 '홍'자만 비교하여 같다고 인식하기 때문에 참값이 나온다. 이때 거짓값이 나오고 완전히 똑같아야만 참값이 나오게 하려면 SET EXACT ON으로 설정하면 된다.

만약 '홍'자만 입력했을 때 위의 공백 제거명령을 사용하면 홍씨 중 첫번째를 찾기 때문에 첫글자의 입력으로도 쉽게 찾을 수 있게 된다.

**(55) SEEK ans**

ans의 변수를 찾는다. SEEK는 처음의 레코드로부터 차례로 비교해 가며 찾는 것이 아니라 인덱스된 키의 트리구조에서 찾기 때문에 고속으로 찾아낸다. 이 명령은 변수명으로 찾지않고 직접 찾고자 하는 데이터를 주어야 하므로 이 명령대신 FIND라는 명령을 쓰고자 한다면 다음과 같이 &를 사용하면 될 것이다.

```
FIND &ans
```

여기서 한가지 명령을 더 알아보자. 데이터를 검색하는 명령으로 LOCATE라는 명령이 있다. 이는 인덱스화일이 필요하지 않으며, 찾는 방법은 첫번째 레코드로부터 비교하여 찾기 때문에 찾는 속도가 느리다. 위의 명령을 LOCATE FOR name=ans .OR. comp=ans로 할 수 있다.

**(56) IF .NOT. FOUND()**

FOUND()라는 명령은 FIND, SEEK 등에 의해서 데이터검색을 했을 경우 찾으면 .T. 값을 주고 찾지 못하면 맨 마지막 레코드보다 하나 더 내려간 End-Of-File에 위치하며 .F. 값을 준다. 그러므로 이 명령은 IF EOF()라는 명령으로 대신 사용할 수 있다. 못찾았을 경우는 EOF에 가있으며 EOF() 함수값을 .T.로 정의하고 있다.

**(57) GO TOP**

첫번째 레코드로 가라는 명령이다. 인덱스되어 있지 않는 화일이라면 레코드번호 1에 위치해 있을 것이고 인덱스화일과 같이 열려 있을 경우는 마스터 키에 의해 배열된 첫번째에 위치하게 된다.

여기서는 찾지 못했을 경우에 당연히 첫번째 레코드에 가서 처음부터 보여준다. 만약 ans값에 아무값도 주지 않고 리턴을 쳤을 경우는 ans=LTRIM(RTRIM(ans))에 의해 null값이 " "으로 아무 것도 없는 값이 되고 SEEK로 찾을 때 무엇이든지 다 찾은 것으로 간주하여 첫번째 레코드에 위치하게 된다.

**(60) @4, 8 CLEAR TO 16, 73**

박스 그리는 명령과 같이 지정한 점으로 박스를 그리는 대신 화면상의 박스 부분의 내부만을 지우는 명령이다.

**(62-186) DO WHILE .T. ... ENDDO**

한 레코드씩 화면에 보여주며 작업을 계속하기 위해 LOOP를 사용한다.

**(63-69) mname = name**

필드의 내용을 모두 변수로 바꾸어 저장한다. 이것은 데이터 열람 및 삭제만 하는 경우는 필요가 없겠으나 수정시 다시한번 확인하여서 맞을 경우만 필드에 재저장하기 위해 변수를 임시로 사용하기 때문이다.

**(70) SET COLOR TO W/N**

화면의 글씨를 흐리게 하며 GET에 의해 디스플레이되는 부분은 계속 W/N+가 되도록 지정하지 않는다.

**(72) DO SCREEN WITH 5**

SCREEN.PRG를 부른다. 여기서는 파라미터값을 5로 주어서 입력하는 화면보다 2칸 위에 디스플레이되게 했다. (SCREEN.PRG는 다음 호에 소개하게 될 것이므로 여기서는 우선 명령어를 이해하기 바란다) 파라미터 기법을 이용하지 않고 서브프로그램을 부르기전에 pline이라는 변수를 미리 지정해 주어도 상관없으나 보다 편리한

방법은 파라미터를 많이 사용하는 것인데 이는 하나의 화일을 여러번 사용하는데에 좋은 효과를 얻을 수 있다.

(75) CLEAR GETS

DO SCREEN에서 사용한 GET 명령은 다음에 READ가 나오면 모두 새로 입력을 받아 들이게 된다. GET은 입력받을 준비만하고 READ에 의해 입력을 받을 수 있다. 입력받을 준비를 하고 있는 GET을 취소하는 명령이 CLEAR GETS이다. 이렇게 하여 조회하는 부분에서는 SCREEN.PRG를 이용하여 디스플레이만 하는 효과를 갖는다.

(76-79)

키입력을 위한 안내문을 디스플레이한다. 전후 열람을 위해서는 숫자를 입력해야 하고 그 범위는 −99에서 999까지이다. F9키를 누르면 −1을 암시하고 F10을 누르면 1을 입력한 것과 마찬가지가 된다.

F3을 누르면 앞부분의 전화번호를 걸어주고 F4를 누르면 뒤의 전화번호를 걸어준다. C를 누르면 현재의 레코드를 수정할 수 있으며, D를 누르면 현재의 레코드를 삭제할 수 있다. Q를 누르면 화면이 끝나고 메인메뉴로 돌아간다.

(81-92) DO WHILE .T. ... ENDDO

디스플레이되어 있는 내용을 계속 열람할 것인가 수정 또는 삭제할 것인가 또는 디스플레이된 전화번호로 전화걸 것인가 하는 명령을 체크하여 잘못 입력된 경우는 다음의 명령을 다시 받기위해 LOOP를 사용한다.

(82) ans = [ ]

999까지 사용하기 위해 3자리로 정해 놓는다.

(83) @ 20,63 GET ans PICT [!!!]

3자리 모두 알파벳인 경우에 소문자나 대문자를 모두 대문자로만 받아 들인다.

(84) READ

앞의 GET에 의해 변수를 새로 입력받는다. 특히 SCREEN.PRG에서 나오는 GET은 CLEAR GETS이란 명령으로 입력을 준비하는 GET을 잊어버리게 했다. 그러므로 여기서는 ans값 하나만 입력으로 받아들이게 된다.

(85) ans = LTRI(RTRI(ans))

ans 변수의 앞뒤의 공백을 제거한다.

(86) IF VAL(ans) = 0 .AND. .NOT. ISALPHA (ans) .OR. LEN(ans) = 0

ans변수가 알파벳이 아니면서 숫자가 0인 경우나 변수의 길이가 0인 경우. (0을 입력하였거나 아무 것도 입력하지 않았을 경우) VAL( )은 Character를 변수로 바꾸는 함수이다.

(89) IF ans = [C] .OR. ans = [D] .OR. ans = [TE] .OR. ans = [Q];
.OR. VAL(ans) <= 999
.AND. VAL(ans) >= −99

논리는 .NOT. .AND. .OR.의 순서으로 진행되므로 [C], [D], [Q], [−99 ~999(0은 제외)], [TE]인 경우

(93) @ 4, 60 SAY "　　　"

[첫번째] 또는 [마지막]이란 메시지가 있을 경우 그 부분을 지운다.

(94) SET COLOR TO W/N+

밝은색 글씨로 재지정

(95-185) DO CASE...CASE...ENDCASE

명령 선택에 대해 실행하는 부분

(96) CASE and <= [999]

숫자인 경우로 위에서 잘못된 명령이 Exit가 되지 않으므로 999보다 작은 경우는 숫자가 된다.

숫자만큼 레코드를 이동시키고자 할 때 사용된다.

(97) SKIP VAL(ans)

숫자만큼 레코드를 전후로 이동시킨다.

(98-106) DOCASE-ENDCASE

에러를 대비한 현재 레코드의 위치를 이동시킨다.

(99) CASE BOF( )

현재의 레코드가 첫번째 레코드보다 한칸 위에 있을 때 첫번째 레코드로 이동하게 된다. BOF에 위치해 있을 때 skip −n이 나오면 에러가 발생한다.

(101-102) SKIP SKIP −1

BOF에서 SKIP하면 두번째 레코드로 이동을 하고 여기서 다시 SKIP −1을 하면 첫번째 레코드로 이동한다. 여기서 또다시 SKIP −1을 하면 BOF로 간다. BOF에서 첫번째 레코드로 이동하기 위해서는 SKIP과 SKIP −1을 각각 한번씩 해야 한다. 여기서는 간단히 GO TOP이라는 명령으로 가능하지만 BOF와 첫번째 레코드와 관계를 알려주기 위해 두번의 SKIP 명령을 사용한다.

(103) CASE EOF( )

현재의 레코드가 마지막 레코드보다 한칸 아래 EOF에 있을 때 마지막 레코드로 가게 한다. EOF()에 위치해 있을 경우 SKIP 명령이 나오면 에러가 발생된다.

(105) SKIP −1

EOF에서 마지막 레코드로 이동시키기 위한 명령이다. 여기서 GO TOP 명령으로 해도 된다.

(108) CASE ans = [C]

C를 입력하여 현재의 레코드를 수정하고자 했을 경우

(111) @ 2, 52 SAY "수정"

수정모드임을 알리는 메시지를 디스플레이한다.

(113-114)

아래의 안내문을 지우고 새로운 안내문을 디스플레이한다.

(117) DO SCREEN WITH 5

SCREEN.PRG를 부르고 각변수의 내용을 디스플레이한다.

(119) READ

SCREEN.PRG에 의해 디스플레이되고 입력준비가 된 것을 받아들인다.

(112-126)

수정한 내용을 확인한다. 옳으면 'Y', 다시 수정해야 할 경우는 'N'을 선택하고 'Y'와 'N'외에는 받아들이지 않는다.

(120-127) @ 20, 25 SAY SPACE(35)

안내문을 지운다.

(128) IF ans = [Y]

수정한 내용을 확인하였을 경우

**(129-131) REPLACE name with mname, …**

현재의 레코드의 필드를 변수의 내용으로 바꾼다. 역시 인덱스화일도 함께 열려 있으므로 인덱스화일도 함께 재배열된다.

**(133) @ 2,52 SAY "열 람"**

수정이 끝났으므로 다시 '열 람'임을 표시해준다.

**(138) CASE ans=[D]**

[D]를 눌러 삭제하고자 했을 때 삭제 여부를 물어보고 입력을 받는다.

**(141-145) DO WHILE…ENDDO**

삭제확인 여부를 묻는다.

**(147) IF ans=[Y]**

삭제확인이 되었을 경우

**(148) DELETE**

현재의 레코드를 삭제한다. 레코드를 완전히 삭제하는 것이 아니고 레코드의 첫부분에 삭제 표시만을 해둔다. 메인프로그램에서 SET DELETE ON으로 환경을 설정해 두었기 때문에 삭제표시된 것은 보이지 않고 SKIP, SEEK, LOCATE 등의 명령을 사용해도 없는 레코드로 간주한다. 그러나 SET DELETE OFF 상태에서는 삭제표시 하지 않은 레코드와 똑같이 나오고 단지 삭제표시 만을 보여준다.

**(149) SKIP**

삭제표시를 하였지만 레코드가 이동하지 않으면 SET DELETE ON 상태라도 현재의 레코드에 위치하고 있다. 그러므로 방금의 데이터가 지워진 것처럼 하기 위하여 지운 레코드에서 다음 레코드로 이동한다.

**(150-152) IF EOF( )…ENDIF**

현 레코드가 마지막 레코드이면 EOF로 가서 다음에 SKIP명령이 나오면 에러가 발생하게 되므로 삭제표시가 없는 레코드중 마지막 레코드로 가게 한다.

**(155) CASE ans=[TE]**

F3 또는 F4를 눌렀을 경우. (전화걸기에 선택했을 경우)

**(156) IF ans=[TE1]**

F3을 눌렀을 경우. (전화 번호 1을 선택했을 경우)

**(157) PHONE=TRIM(TEL1)**

PHONE이라는 변수에 현재 레코드의 TEL1이라는 필드값을 넣는다.

**(164-168) DO WHILE…ENDDO**

전화번호를 다시 디스플레이 해주며 확인한다.

**(170) IF ans=[Y]**

전화걸기 확인을 끝내고 전화를 걸도록 선택했을 경우

**(171) SET ALTERNATE TO TEMP.TEL**

@ …SAY …GET에 의하지 않고 디스플레이되는 모든 것은 TEMP.TEL이라는 화일에 저장하기 위해 화일을 연다.

**(172) SET CONSOLE OFF**

모니터에 디스플레이되는 것을 중지한다.

**(173) SET ALTERNATE ON**

지금부터 @ …SAY …GET에 의하지 않고 디스플레이되는 모든 것은 TEMP.TEL이라는 화일에 저장을 한다.

**(176) SET ALTERNATE OFF**

화일에 저장하는 것을 중지한다.

**(177) SET ALTERNATE TO**

ALTERNATE 화일을 닫는다.

**(178) SET CONSOLE ON**

화면 디스플레이 기능을 되살린다.

**(179) ! TYPE TEMP.TEL〉COM1**

현재 TEMP.DAT라는 화일 안에는 [atdt]라는 글자와 전화번호 그리고 하나의 리턴 값이 들어가 있는 작은 화일이 있다. 이 화일을 모뎀이 장착되어 있는 통신포트로 보내면 모뎀은 다이얼 톤을 발생시켜 전화를 걸어준다. 만약 기계식 전화국의 전화라면 프로그램에 [atdt] 대신에 [atdp]라고 해야 한다. 모뎀과 전화 컴퓨터와의 연결은 그림 1과 같다.

**(180) ERASE TEMP.TEL**

SET ALTERNATE로 만들었던 화일로 전화를 건후에는 화일을 지운다.

**(181) @ 20, 20 SAY "수화기를 든후에 아무키나 누르시오"**

전화를 거는 소리가 멈추었을 경우 수화기를 들고 아무키나 누르게 한다.

**(182) WAIT[ ]**

아무키나 누름으로써 프로그램이 계속된다. 여기서 잠시 멈추게 한 이유는 다음의 명령이 모뎀의 내부를 전화라인과 끊어서 전화에서 잡음이 들리지 않도록 해주는데 전화가 완전히 걸리지 않는 상태에서 모뎀 내부를 라인과 끊으면 전화가 걸리다 도중에 끊어지게 되므로 전화가 완전히 걸리게 되는 경우(상대방 전화에 신호가 가는 경우)까지 기다리게 해주어야 한다.

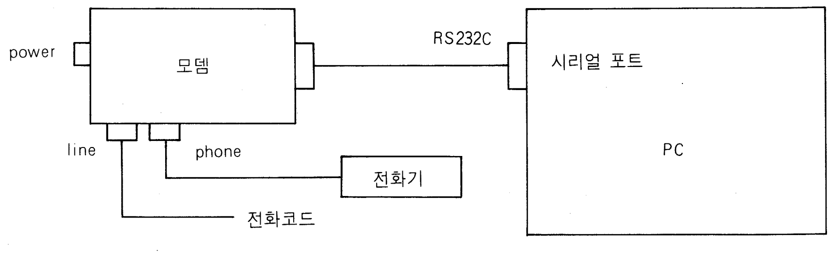
아무키나 누르기 전에 반드시 수화기를 들어야 한다. 수화기를 들지않고 아무키를 누르면 곧바로 다음 명령에 의해 모뎀은 전화를 끊어 버리게 된다.

**(183) ! TYPE HANGUP.TEL〉COM1**

모뎀과 라인이 끊어지게 하는 명령이 들어 있는 화일을 통신포트로 보내준다. 모뎀에서는 전화라인을 끊어 줌으로써 전화기에서는 잡음이 들리지 않게 된다.

HANGUP. TEL을 만드는 방법은 다음과 같다.

〔그림 1〕



```
C:\>COPY CON HANGUP.TEL
ATH ^Z <RETURN>
      1 File(s) copied
```

**(190-193) SET FUNCTION 3 TO [ ]…** 지정한 펑션키를 해제한다.

**(186) CASE ans=[Q]** 열람 중지를 위해 'Q'를 눌렀을 경우

**(194) RETURN** 모프로그램으로 복귀한다.

```
1 * Program..: READ.prg
2 * Author...: Bahk Seonghyeon , Han Kyumyun
3 * Date.....: 1989. 4. 5.
4 * Notes....: 전화 번호부 읽기 프로그램
5 * Notice...: Called from MAIN.prg
6
7 SET FUNCTION 3  TO [TE1]
8 SET FUNCTION 4  TO [TE2]
9 SET FUNCTION 9  TO [-1;]
10 SET FUNCTION 10 TO [1;]
11 SET COLOR TO W/N+
12 CLEAR
13 TEXT
14 ╔══════════════════════════════╗
15 ║     전 화   번 호 부   열 람     ║
16 ╟──────────────────────────────╢
17 ║                              ║
18 ║                              ║
19 ║                              ║
20 ║     성 명 순 ( 1 )   회 사 순 ( 2 ) ║
21 ║                              ║
22 ║                              ║
23 ║                              ║
24 ║                              ║
25 ║                              ║
26 ║                              ║
27 ║                              ║
28 ║                              ║
29 ║                              ║
30 ╟──────────────────────────────╢
31 ║                              ║
32 ║                              ║
33 ║                              ║
34 ║                              ║
35 ╚══════════════════════════════╝
36 ENDTEXT
37
38 ndx1=[이름]
39 ndx2=[회사]
40
41 SET COLOR TO W/N,W/N+
42 ord=[?]
43 DO WHILE .NOT. ord$[12]
44 ¦   ord=[?]
45 ¦   @ 7,60 GET ord PICTURE [9]
46 ¦   READ
47 ENDDO
48
49 SET ORDER TO &ord
50 ans=SPACE(20)
51 @ 11,21 SAY [열람  하고자  하는  ]+ndx&ord.+[를  입력  하시오.]
52 @ 14,26 GET ans
53 READ
54 ans=LTRIM(RTRIM(ans))
55 SEEK ans
56 IF .NOT. FOUND()
57 ¦   GO TOP
58 ENDIF
59
60 @ 4,8 CLEAR TO 16,73
61
62 DO WHILE .T.
63 ¦   mname  = name
64 ¦   mcomp  = comp
65 ¦   mjik   = jik
66 ¦   maddr  = addr
67 ¦   mwoo   = woo
68 ¦   mtel1  = tel1
69 ¦   mtel2  = tel2
70 ¦   SET COLOR TO W/N,W/N+
71 ¦   *--------------*
72 ¦   DO SCREEN WITH 5
73 ¦   *--------------*
74 ¦   CLEAR GETS
75 ¦                                           F9=-1 , F10=1 ) "
76 ¦   @ 18,9  SAY "전 후 열 람  --------->  숫자 (-99 --- 999 ,
77 ¦   @ 19,9  SAY "전 화 걸 기  --------->  < F3 > , < F4 > "
78 ¦   @ 20,9  SAY "수정 , 삭제  --------->  < C >  ,  < D > "
79 ¦   @ 21,9  SAY "열 람 중 지  --------->  < Q >            "
80 ¦
81 ¦   DO WHILE .T.
82 ¦   ¦   ans=[   ]
83 ¦   ¦   @ 20,63 GET ans PICT [!!!]
84 ¦   ¦   READ
85 ¦   ¦   ans=LTRI(RTRI(ans))
86 ¦   ¦   IF VAL(ans)=0 .AND. .NOT. ISALPHA(ans) .OR. LEN(ans)=0
87 ¦   ¦   ¦   LOOP
88 ¦   ¦   ENDIF
89 ¦   ¦   IF ans=[C] .OR. ans=[D] .OR. ans=[TE] .OR. ans=[Q] ;
89 ¦   ¦           .OR. VAL(ans)<= 999 .AND. VAL(ans) >= -99
```

```
¦    ¦    ¦    EXIT
¦    ¦    ENDIF
¦    ENDDO
¦    @ 4,60 SAY "                    "
¦    SET COLOR TO W/N+
¦    DO CASE
¦    ¦    CASE ans <= [999]
¦    ¦    ¦    SKIP VAL(ans)
¦    ¦    ¦    DO CASE
¦    ¦    ¦    ¦    CASE BOF()
¦    ¦    ¦    ¦    ¦    @  4,60 SAY " 첫 번 째 "
¦    ¦    ¦    ¦    ¦    SKIP
¦    ¦    ¦    ¦    ¦    SKIP -1
¦    ¦    ¦    ¦    CASE EOF()
¦    ¦    ¦    ¦    ¦    @  4,60 SAY " 마 지 막 "
¦    ¦    ¦    ¦    ¦    SKIP -1
¦    ¦    ¦    ENDCASE
¦    ¦    ¦
¦    ¦    CASE ans=[C]
¦    ¦    ¦
¦    ¦    ¦    SET COLOR TO W/N+
¦    ¦    ¦    @  2,52 SAY "수  정"
¦    ¦    ¦    SET COLOR TO W/N,W/N+
¦    ¦    ¦    @ 18,8 CLEAR TO 21,73
¦    ¦    ¦    @ 20,25 SAY " 위의  내용을  수정  하십시오."
¦    ¦    ¦    DO WHILE .T.
¦    ¦    ¦    ¦    *--------------*
¦    ¦    ¦    ¦    DO SCREEN WITH 5
¦    ¦    ¦    ¦    *--------------*
¦    ¦    ¦    ¦    READ
¦    ¦    ¦    ¦    @ 20,25 SAY SPACE(35)
¦    ¦    ¦    ¦    ans=[?]
¦    ¦    ¦    ¦    DO WHILE .NOT. ans$[YN]
¦    ¦    ¦    ¦    ¦    ans=[?]              GET ans PICT [!]
¦    ¦    ¦    ¦    ¦    @ 20 ,25 say "수정 확인 ( Yes / No ) "
¦    ¦    ¦    ¦    ¦    READ
¦    ¦    ¦    ¦    ENDDO
¦    ¦    ¦    ¦    @ 20,25 SAY SPACE(35)
¦    ¦    ¦    ¦    IF ans=[Y]                    jik with mjik
¦    ¦    ¦    ¦    ¦    REPLACE name with mname, comp with mcomp,
¦    ¦    ¦    ¦    ¦    REPLACE addr with maddr, woo  with mwoo
¦    ¦    ¦    ¦    ¦    REPLACE tel1 with mtel1, tel2 with mtel2
¦    ¦    ¦    ¦    ¦    SET COLOR TO W/N+
¦    ¦    ¦    ¦    ¦    @  2,52 SAY "열  람"
¦    ¦    ¦    ¦    ¦    EXIT
¦    ¦    ¦    ¦    ENDIF
¦    ¦    ¦    ENDDO
¦    ¦    ¦
¦    ¦    CASE ans=[D]
¦    ¦    ¦    @ 18,8 CLEA TO 21,73
¦    ¦    ¦    ans=[?]
¦    ¦    ¦    DO WHILE .NOT. ans$[YN]
¦    ¦    ¦    ¦    ans=[?]
```

```
¦    ¦    ¦    ¦    @ 20 ,25 say "삭제 확인 ( Yes / No ) "
¦    ¦    ¦    ¦    READ              GET ans PICT [!]
¦    ¦    ¦    ENDDO
¦    ¦    ¦    @ 20,25 SAY "                              "
¦    ¦    ¦    IF ans=[Y]
¦    ¦    ¦    ¦    DELETE
¦    ¦    ¦    ¦    SKIP
¦    ¦    ¦    ¦    IF EOF()
¦    ¦    ¦    ¦    ¦    GO BOTTOM
¦    ¦    ¦    ¦    ENDIF
¦    ¦    ¦    ENDIF
¦    ¦    ¦
¦    ¦    CASE ans=[TE]
¦    ¦    ¦    IF ans=[TE1]
¦    ¦    ¦    ¦    PHONE=TRIM(TEL1)
¦    ¦    ¦    ELSE
¦    ¦    ¦    ¦    PHONE=TRIM(TEL2)
¦    ¦    ¦    ENDIF
¦    ¦    ¦
¦    ¦    ¦    @ 18,8 CLEA TO 21,73
¦    ¦    ¦    ans=[?]
¦    ¦    ¦    DO WHILE .NOT. ans$[YN]
¦    ¦    ¦    ¦    ans=[?]              >> ( Yes/No ) ' ;
¦    ¦    ¦    ¦    @ 20 ,10 SAY '전화 번호 확인 : << '+phone+'
¦    ¦    ¦    ¦              GET ans PICT [!]
¦    ¦    ¦    ¦    READ
¦    ¦    ¦    ENDDO
¦    ¦    ¦    @ 20,10 SAY SPACE(60)
¦    ¦    ¦    IF ans=[Y]
¦    ¦    ¦    ¦    SET ALTERNATE TO TEMP.TEL
¦    ¦    ¦    ¦    SET CONSOLE OFF
¦    ¦    ¦    ¦    SET ALTERNATE ON
¦    ¦    ¦    ¦    ?? 'atdt '+phone
¦    ¦    ¦    ¦    ?
¦    ¦    ¦    ¦    SET ALTERNATE OFF
¦    ¦    ¦    ¦    SET ALTERNATE TO
¦    ¦    ¦    ¦    SET CONSOLE ON
¦    ¦    ¦    ¦    ! TYPE TEMP.TEL > COM1
¦    ¦    ¦    ¦    ERASE TEMP.TEL
¦    ¦    ¦    ¦    @ 20,20 SAY  "수화기를 든후에 아무키나 누르시오 "
¦    ¦    ¦    ¦    WAIT []
¦    ¦    ¦    ¦    ! TYPE HANGUP.TEL > COM1
¦    ¦    ¦    ¦    @ 20,10 SAY SPACE(60)
¦    ¦    ¦    ENDIF
¦    ¦    CASE ans=[Q]
¦    ¦    ¦    EXIT
¦    ENDCASE
ENDDO
SET FUNCTION 3  TO []
SET FUNCTION 4  TO []
SET FUNCTION 9  TO []
SET FUNCTION 10 TO []
RETURN
```

# 명함관리 프로그램

이기성 / 장왕교재 연구원 상무

지금까지 7회에 걸쳐 베이직언어를 소개했다(표 1 참조). 한정된 지면으로 다방면의 기능을 갖고 있는 베이직을 고루고루 소개하기란 사실 무리이지만 본 지면에 실린 내용만 국한시키지 말고 (독자들이 본 지면을 계기로) 관련 명령어 숙지 및 그 응용을 해주어야 할 것이다.

필자는 이러한 확장된 학습효과를 기대하여 표 1에서 보듯이 5회째 부터는 간단한 실무 프로그램을 소개해 왔다. 단순한 명령어 해설에 그치지 않고 실무 프로그램 작성 능력까지도 소개하고픈 필자의 성급한 마음이니 양해해주길 바란다. (사실 대부분의 명령어 해설 수준에 그치는 글을 읽고는 실제 유용하게 쓰일 수 있는 프로그램을 작성하기란 여전히 어렵게 된다.)

이번에는 마지막으로 명함을 효율적으로 관리할 수 있는 프로그램을 소개하겠다.

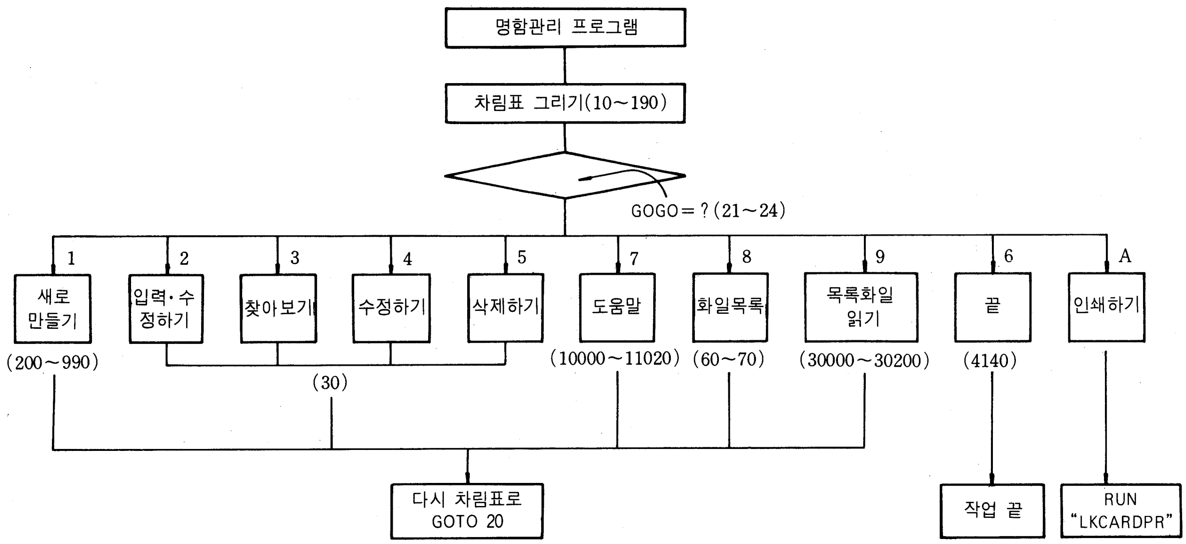
〔표 1〕 지금까지 연재된 내용

| 연 재 | 게 재 호 | 내 용 |
|---|---|---|
| 1회 | 1988, 12 | 직접모드와 간접모드 |
| 2회 | 1989, 1 | 순차화일과 랜덤화일 |
| 3회 | 2 | 그래픽 |
| 4회 | 3 | 사운드와 프로그램 분기 |
| 5회 | 4 | 레코드판 관리프로그램 |
| 6회 | 5 | 교육용 프로그램 |
| 7회 | 6 | 명함관리 프로그램 |

〔표 2〕 레코드의 항목 지정

| 항 목 | 길이(단위 : 바이트) |
|---|---|
| ① 이 름 | 20 |
| ② 지역코드 | 4 |
| ③ 전화번호-1 | 10 |
| ④ 구내번호 | 4 |
| ⑤ 전화번호-2 | 12 |
| ⑥ 회사이름 | 25 |
| ⑦ 근무부서 | 20 |
| ⑧ 비 고 | 16 |

〔그림 1〕 전체적인 프로그램 분기




※ ( )속의 숫자는 문번호임

명함관리에 필요한 레코드의 길이는 최고 111바이트로 설정하고 한 레코드는 다시 8개의 항목으로 나누었다(표 2 참조). 메인 프로그램은 LKCARD.BAS(리스트 1)으로 이곳에서 카드화일을 새로 만들고 데이터를 입력 및 수정 또는 찾아보기의 작업을 하게 된다. 그리고 프린터에 인쇄하는 프로그램은 LKCARDPR.BAS(리스트 2)로, 메인 프로그램에서 'A'를 선택하면 자동으로 인쇄루틴으로 들어가게 된다.

참고로, 이 프로그램은 IBM XT/AT 기종에 맞도록 베이직으로 작성되었으나, 애플 CP/M에서도 한글을 화면에 사용할 수 있는 기종은 커서위치 지정명령과 화면 지우기 명령, 한글 시작코드만 수정하면 이 프로그램을 그대로 사용할 수 있음을 말해 둔다.

## 프로그램 실행

IBM PC XT나 AT에 전원을 ON 시키면 잠시후 A> 프롬프트가 나온다. 이때 BASIC.EXE 화일이 들어 있는 디스켓으로 바꿔 넣고 A>BASIC을 입력하면 베이직 인터프리터 모드로 들어간다. 완성된 명함관리 프로그램일 경우 RUN "LKCARD"하면 시작화면이 나온다(화면 1 참조).

리스트 1의 LKCARD.BAS를 보면 100행에서 190행까지 메뉴를 그리는 부분이 나와 있다. 여기서의 항목 선택은 21행에서 다음과 같이 행해진다. 즉

```
21 LOCATE 22,3:PRINT"어느 번호를 원하십니까?
        [0-9,A]";:GOGO$=INPUT$(1)
22 IF GOGO$="a" OR GOGO$="A" THEN PRINT
        "잠간만...":RUN "lkcardpr"
24 GOGO=VAL(GOGO$): ON GOGO GOSUB
        200,30,30,30,30,4140,10000,60,30000
```

INPUT$(1)으로 한 문자(문자 형태임에 주목하라)를 입력받는다. 만일 입력받은 문자가 'a'이거나 'A', 즉 프린터로 인쇄하기를 원했을 때는 22행의 RUN "lkcardpr"에 의해 프로그램의 제어는 리스트 2의 LKCARDPR로 넘어가게 되고, 그외의 경우는 우선 VAL 함수를 이용해 입력문자를 수치 형태로 변환한 후 ON~GOGO~GOSUB 문으로 제각기 분기를 시킨다. 전반적인 프로그램의 분기구조는 그림 1에 보였으며, 크게 시작 차림표 부분과 입력 및 찾아보기 루틴으로 나누어짐을 볼 수 있다.

프로그램을 처음에 이용할 때는 명

〔화면 1〕 명함관리 시작 차림표

```
---------------------------------------------------------------------

             ***▒     명  함    관  리     ▒***

만듬 0.9                                          장왕사 전산실

#=====================================================================#

       1. 카드 새로 만들기
       2. 카드 입력, 수정하기
       3. 자료 찾아보기
       4. 자료 수정하기
       5. 자료 삭제하기
       6. 끝
       7. 도움말
       8. 디스켓 화일 목록 살피기
       9. 목록 (.SKY) 화일 읽기
       A. 프린터로 인쇄하기

           ..... 어느 번호를 원하십니까  [ 1 - 9,A ]

---------------------------------------------------------------------
```

〔화면 2〕 찾아보기 화면

```
---------------------------------------------------------------------

          ▒▒▒   명함 (전화 번호) 찾기    ▒▒▒

    1. 레코드 번호로 찾기
    2. 연속하여 보여 주기
    3. 항목별로 찾아 보기 ( =, <, > )
    4. 키(1번) 항목으로 찾기

    0. 시작 차림표로 돌아가기.

              ..... 어느 번호  : 3

---------------------------------------------------------------------
```

함카드 화일을 먼저 만들어야 하므로 시작 차림표에서 '1'을 선택해야 한다. 여기서 한가지 주의해야 할 점은 '처음 이용'이란 말은 매번 컴퓨터 작업을 할 때의 처음이 아니라 이 프로그램을 이용하여 명함관리 작업을 처음으로 시작할 때를 의미한다는 것이다. 왜냐하면 이 루틴은 명함화일의 구조를 만들어 디스켓에 'LKCARD.SIF'라는 이름으로 기록하기 때문에 이 화일을 만든 후 다시 이 루틴을 실행시키면 기존의 데이터들을 모두 지우고 다시 같은 이름의 화일구조를 만들어 버리기 때문이다. (이를 감안하여 실행중 이 루틴을 부르면 조심하라는 메시지를 출력하게 하였다.) 화일의 구조를 만들고 기억시키는 부분은 다음과 같다.

```
712 INFILENAME$="lkcard"+".sif"
```

구조화일 이름

```
720 OPEN INFILENAME$ FOR OUTPUT AS 1
722      PRINT #1, R.GILI
724      PRINT #1, R.HANGMOKSOO
```

화일을 열고 레코드 길이와 항목수를 기록

```
730 FOR J=1 TO 8
740      PRINT #1, TITLE$(J)
750 NEXT
```

8개의 항목이름을 기록

```
830 FOR J=1 TO 8
840      PRINT #1, ENTRY$(J)
850 NEXT
```

각 항목당 차지하는 바이트수를 기록

이 작업이 끝나면 그림 2의 (a)처럼 8개의 항목 타이틀이 지정되고 후에 데이터를 입력하면 (b)의 상태로 기억을 하게 된다.

다음은 '자료입력하기' 순서가 될 것이다. 프로그램의 30번 행부터 이에 해당하는데 이 루틴은 데이터의 입력과 수정 및 삭제, 찾아보기 등 모두를 전담하고 있으며, 화살표 키를 이용한 편리한 수정과 찾아볼 때도 여러 상황(레코드 번호로 찾기, 연속해서 보여주기, 항목별로 찾기, 키〈1번〉 항목으로 찾기, 내용의 첫자만 알 때)을 고려하였기 때문에 이해하는데 조금은 힘든 점이 없지 않을 것이다. 그러나 실무 프로그램은 누구든지 사용하기 쉽게 작성되어야 하고 그렇게 하기 위해서는 프로그래머가 그만큼 더 수고를 해 주어야 한다는 것을 알아두기 바란다. 그림 3에 데이터의 입력 및 찾아보기, 수정하기, 삭제하기의 흐름 구성을 보였으니 참고하기 바란다. 입력루틴중, 3085행에서는 3300행으로 가서 한 항목씩 입력을 받고, 3370행에서는 바이트를 입력받아 하나의 항목이 완성되면 다시 3380행에서 RETURN으로 3086행으로 보내준다. 3075행부터 3115행까지는 WHILE~WEND의 루프로 입력

〔그림 2〕 명함화일의 구조와 데이터

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 이름 | 지역코드 | 전화번호-1 | 구내번호 | 전화번호-2 | 회사이름 | 근무부서 | 비고 |

(a) 명함구조(LKCARD.SIF) 화일

1번 레코드

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 권옥희 | 02 | 718—9807/8 | 69 | 718—0005 | 흥국생명 | 영업부 | 주임 |

2번 레코드

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 김영주 | 053 | 763—9803 | | 476—3479 | 신구대학교 | 인쇄과 | 출판전공 |

(b) 명함 데이터(LKCARD.SOK) 화일

〔화면 3〕 항목별로 찾아보기

```
1] 이름                  2] 지역코드              3] 전화번호-1
4] 구내번호              5] 전화번호-2            6] 회사 이름
7] 근무 부서             8] 비고

=================================================================
찾아볼 항목 번호는 [1--8] <┘  : 6

 찾는 내용의 첫자만 알 때는 +를 누르세요. 아니면 그냥 <┘ 을    +
 찾을 내용의 첫자는 ==> 홍



• 찾았읍니다.    # 1

보시겠읍니까 (1= 봄,11= 연속봄,0= 안봄,7= 프린터로, 77= 프린터로 연속봄)<┘  1
```

```
.........리턴을 누르면 계속 . ( 0= 그만 보고 차림표로 )
=================================================================
 1   이름              :  권옥희
 2   지역코드          :  02
 3   전화번호-1        :  718-9807/8
 4   구내번호          :  69
 5   전화번호-2        :  718-0005
 6   회사 이름         :  흥국생명
 7   근무 부서         :  영업부
 8   비고              :  주임
-----------------------------------------------------------------

                                  # 1

-----------------------------------------------------------------
```

루틴의 핵심을 이루는 곳이다.

```
3075 WHILE L>0 AND L<=R.HANGMOKSOO
3080    V=2+L: LOCATE V,4:PRINT USING
        "###";L;
3082    ENTRYLEN=H(L)
3085    GOSUB 3300  '한 항목 입력받기
3086    '
3090    IF (EOI=1) AND (L=1)
        THEN PRINT CHR$(7): GOTO 3115
3100    IF EOI=1 THEN L=L-1
3105    IF EOI=2 OR EOI=3 THEN L=L+1
3110    IF EOI=4 THEN L=STRINGFIELDS+1
3115 WEND
```

IBM PC XT/AT 호환기종 중에서 삼보컴퓨터의 제품이나 허큘리스 카드를 사용하는 세운상가 컴퓨터에서는 이 루틴이 잘 작동된다. 즉 CHP, NKP, THP, HP한글을 사용하는 컴퓨터에서는 이 프로그램을 사용할 수 있다. IBM 5550(일본에서 주로 사용되고 있다.) 컴퓨터에서는 3071행부터 3074행의 코멘트 명령' '만 지우고 사용하면 된다. 다시말하면 3073행에서 INPUT명령으로 8개 항목의 내용을 모두 받고나서 3195행으로 간 후 디스켓에 기록하는 것이다.

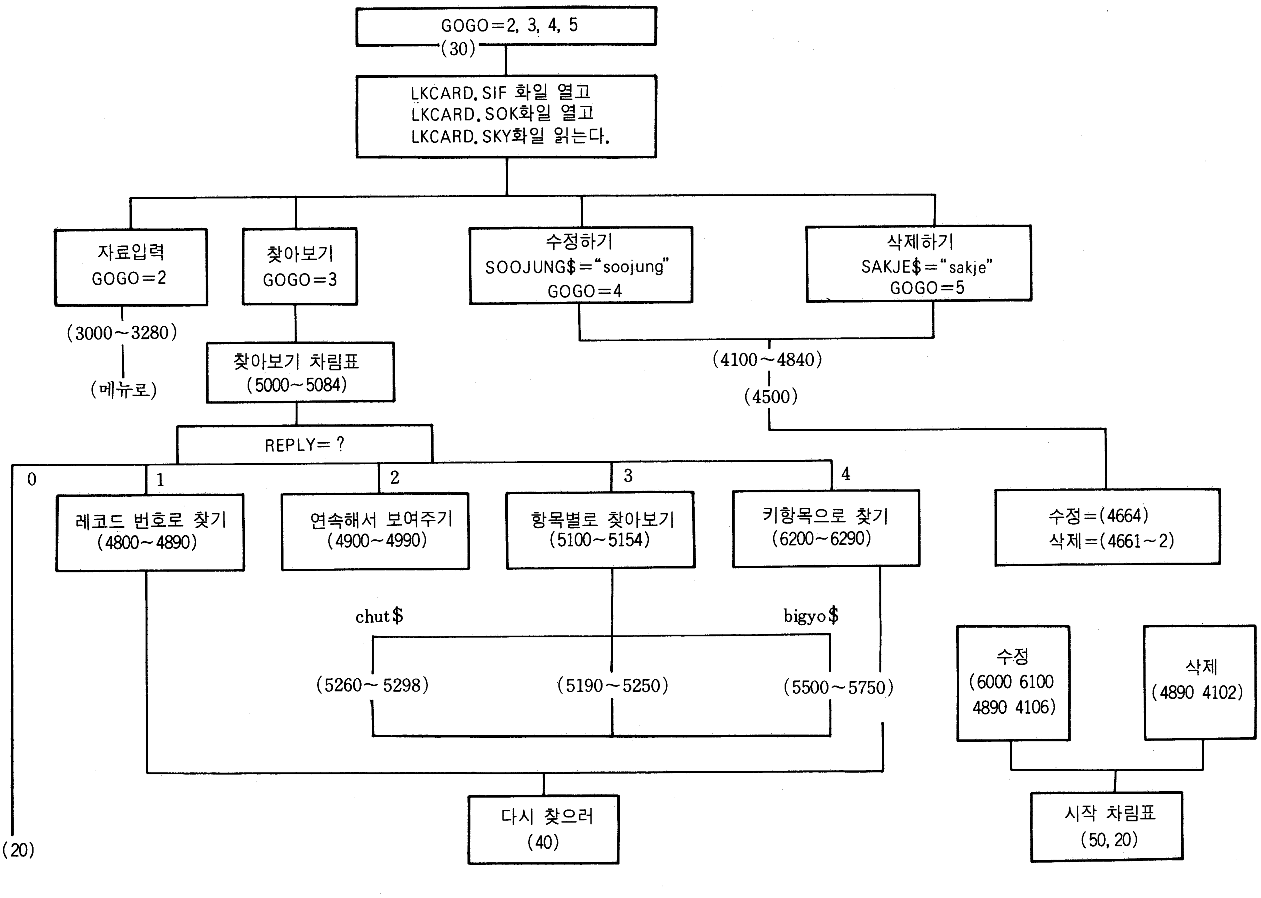
다음은 '자료 찾아보기'이다. 명함(전화번호)찾기는 레코드 번호로 찾을 수도 있고 연속하여 여러개를 살펴볼 수도 있다. 또한 항목별로 원하는 조건을 주어서 찾기도 하고, 이름을 알면 빠르게 찾는 방법도 있다. (화면 2에서 보는 바와 같이 모두 4가지 방법으로 데이터를 찾을 수 있도록 하였다.)

화면 2에서 3번을 선택하면 각 항목별로 비교하여 찾을 수가 있는데, 찾을 내용의 첫자만 알고 있어도 해당 데이터를 골라내 주니까 편리하게 사용할 수 있을 것이다. 화면 3은 6번 항목(회사이름)의 첫 글자가 '홍'(홍국 생명)인 것을 골라 보는 예이다. 〈ENTER〉키를 누르면 해당 내용(첫자가 '홍')과 같은 데이터가 또 있나 계속 찾다가 있으면 '보시겠읍니까'를 다시 묻는다. 끝까지 다 찾았으면 찾기 차림표로 돌아간다.

## 명함(전화번호) 인쇄하기

명함관리의 시작 차림표에서 A(프린터로 인쇄하기)를 고르면 LKCAR-

〔그림 3〕 데이터의 입력, 찾기, 수정, 삭제의 흐름 구성




※( ) 속의 숫자는 문번호

DPR. BAS가 실행되면서 인쇄시작 차림표를 보여 준다. 인쇄는 전부 인쇄할 수도 있고, 원하는 데이터만 골라서 인쇄할 수도 있다(화면 4 참조). 한 레코드의 내용을 인쇄시키는 루틴을 요약해 보면 다음과 같다.

```
20260 IF DISPLAYENTRY > LOF(1)/111
      THEN GOTO 20450
20270 GET #1,DISPLAYENTRY
20280 FOR ITEMNUMBER = 1 TO R.HANGMOKSOO
20290     ENTRY$(ITEMNUMBER)
          =A$(ITEMNUMBER)
20300 NEXT ITEMNUMBER
20340 FOR IREPORT=1 TO 8
20360     IF IREPORT=8 AND HWOISA$<>"1"
          THEN 20430
20362     IF PRINTER$="ON" THEN LPRINT
          ENTRY$(IREPORT);
20370     PRINT ENTRY$(IREPORT);
20430 NEXT IREPORT
20440 PRINT
20442 IF PRINTER$="ON" THEN LPRINT:LPRINT"
........................."
20450 RETURN          '한 레코드 내용을 다 적었음
```

〔화면 4〕 인쇄하기 예

```
-----------------------------------------------------------

1] 이름                 2] 지역코드              3] 전화번호-1
4] 구내번호             5] 전화번호-2            6] 회사 이름
7] 근무 부서            8] 비고

___________________________________________________________
* 찾는(인쇄할) 자료 선택은 몇번 항목 (1--8),  전 자료 다 적기= 0 :  0
인쇄하실 목록의 제목은 : 전체 리스트

프린터를 준비하세요     프린터에=1, 아니면=0    1
8번 항목(비고)도 적으십니까 (적음=1, 안적음=0 (보통 = 0)   1
프린터는 1= 큐닉스(후지쓰) ,  0= 삼보(엡슨) .....  1
```

```
-----------------------------------------------------------

          전체 리스트                        05-26-1989
==============================================================
권옥희    02   718-9807/8 69   718-0005 홍국생명     영업부
          주임
..............................................................
김영주    053  763-9803        476-3479 신구대학교  인쇄과
          출판전공
..............................................................
최혜정    051  66-8116                  부산대학교  국문학과
          한국편집아카데미
..............................................................
--------------------------------------------------------------
3


-----------------------------------------------------------
```

다음은 목록만들기의 루틴을 보자. 20000행부터 21630행까지가 목록만들기에 해당하는 루틴인데, 모든 데이터를 모두 찍을 것인지 아니면 어느 특정 항목을 골라 원하는 조건을 주어서 일부 자료만 골라 찍을 것인지를 20770행에서 ITM$로 입력받는다. 모든 자료를 다 인쇄하려고 '0'을 선택하면 20830행으로 가서 인쇄할 목록의 제목을 입력받는다. 20870행에서 화면에 제목을 보여주고, 20892행의 조건대로 20910, 20920행을 거쳐 21380행으로 간다. 21460행부터 21530행까지에서는 FOR~TO~NEXT문으로 되는 모든 데이터를 인쇄하게 된다. 실제로 한 데이터씩 인쇄하는 루틴은 21490행에서 GOSUB 명령으로 보내는 20260행부터 20450행까지이며, 모두 인쇄하고 나면 21540행에서 21010행으로 보내는 맨 밑줄을 인쇄하고 21300행에서 21370행 사이에서 합계를 인쇄한다. 작업을 마치면 21070 행에서 20행으로 다시 인쇄시작 차림표를 그린다. PC

□ **알림** □ 그동안 7회에 걸쳐 독자 여러분의 성원속에 연재된 '베이직 강좌'가 이번회를 마지막으로 끝마치게 되었습니다. 집필을 맡아 주신 이기성님께 감사드리며, 다음회부터는 다른 내용으로 본란을 꾸미고자 하오니 계속해서 많은 성원을 보내주시길 바랍니다.

〔리스트 1〕 LKCARD. BAS

```
7 '---- created by Lee Kisung
8 KEY OFF
10 '
11 '
12 DIM A$(20) ,TITLE$(20),ENTRY$(20),ENTRY#(20),ITEMSOO(20),ITE
   MTOTAL#(20),KA$(2),KEYNAME$(1000),KEYRECORD(1000),K(60)
14 DIM HANGMOKGILI(20),H(20)
15 ON ERROR GOTO 60000
16 JEIL$="close"
17 LINE1$="#=":FOR I=1 TO 36:LINE1$=LINE1$+"==":NEXT I:LINE1$=L
   INE1$+"=#"
18 UNDERLINE$ = "________________________"
20 GOSUB 100:GOSUB 120            '시작 차림표 만들기
21 LOCATE 22,3:PRINT"             어느 번호를 원하십니까  [ 0 - 9,
   A ] ";:GOGO$=INPUT$(1)
22 IF  GOGO$ = "a"  OR  GOGO$ = "A"  THEN PRINT : PRINT " 잠깐만.
```

```
   ..":RUN "lkcardpr"
24 GOGO=VAL(GOGO$):ON GOGO GOSUB 200,30,30,30,30,4140,10000,60,
   30000
26 GOTO 20
29 END
30 IF JEIL$="close" THEN JEIL$="open": GOSUB 4200
31 SAKJE$= "no": IF GOGO= 5 THEN SAKJE$= "sakje"
32 SOOJUNG$= "no": IF GOGO= 4 THEN SOOJUNG$ ="soojung"
40 ON GOGO GOSUB 200,1000,5000,4100, 4100
42 IF GOGO=3 THEN GOTO 40            '찾아보기를 반복
50 GOTO 20
60 CLS: FILES: PRINT:    PRINT:PRINT:INPUT" ... 리턴을 누르면
   차림표로 갑니다.  ",CHA$
70 RETURN
100 CLS
102 LOCATE 3,18:COLOR 0,7:PRINT"***  명   함    관  리   ***"
    :COLOR 7,0
103 PRINT:PRINT"만듬 0.9";
104 PRINT TAB(60)"장왕사 전산실"
105 PRINT"#=======================================================
====================#"
110 ENTRYLEN = 24
118 RETURN
120 LOCATE 9,9: PRINT"1. 카드 새로 만들기"
122 LOCATE 10,9:PRINT"2. 자료 입력, 수정하기"
124 LOCATE 11,9:PRINT"3. 자료 찾아보기"
125 LOCATE 12,9:PRINT"4. 자료 수정하기 "
128 LOCATE 13,9:PRINT"5. 자료 삭제하기 "
130 LOCATE 14,9:PRINT"6.  끝"
132 LOCATE 15,9:PRINT"7. 도움말 "
138 LOCATE 16,9:PRINT"8. 디스켓 화일 목록 살피기"
139 LOCATE 17,9:PRINT"9. 목록(.SKY) 화일 읽기"
140 LOCATE 18,9:PRINT"A. 프린터로 인쇄하기 "
180 REM
190 RETURN
200 '------ 화 일 만 들 기 ----------------------------------------
202 CLS
210 LOCATE 1,10:PRINT "   명함(전화 번호부) 처음   만들기 "
220 PRINT"=======================================================
====================="
230 PRINT:FILES:PRINT:PLAY "c"
232 PRINT" 새로 화일을 만드십니까?  또 만들면 먼저번의 데이타가 전부"
233 PRINT" 못쓰게되니 조심하세요.";
234 PRINT" (1= 새로 만듬.  0= 그냥 차림표로 감) : ";:A$=INPUT$(1):IF
    A$="1" THEN PRINT:GOTO 240  ELSE BEEP:GOTO 990
240 FILENAME$="lkcard.sok"
260 R.GILI=111
280 R.HANGMOKSOO=8
310 '
320 TITLE$(1)="이름":TITLE$(2)="지역코드":TITLE$(3)="전화번호-1"
330 TITLE$(4)="구내번호":TITLE$(5)="전화번호-2":TITLE$(6)="회사 이름"
340 TITLE$(7)="근무 부서":TITLE$(8)="비고       "
350 CLS
360 PRINT TAB(20);" 항목의 이름 "
370 FOR I = 1 TO R.HANGMOKSOO
380     PRINT I;" : ";TITLE$(I)
390 NEXT
400 PRINT
410 PRINT "                                          〈리턴〉 키를
    누르세요. "
420 INPUT " ---------------------------------------------------
------------------" ,AA$
500 PRINT
520 HANGMOKGILI(1)=20:HANGMOKGILI(2)=4:HANGMOKGILI(3)=10
530 HANGMOKGILI(4)=4:HANGMOKGILI(5)=12:HANGMOKGILI(6)=25
540 HANGMOKGILI(7)=20:HANGMOKGILI(8)=16
550 FOR II=1 TO  8
552    LSET ENTRY$(II)=STR$(HANGMOKGILI(II))
554 NEXT
560 PRINT TAB(5);" 항목의 이름 " ;TAB(30);" 길이 "
562 HANGMOKTTL= 0
570 FOR I = 1 TO  8
572     HANGMOKTTL= HANGMOKTTL + HANGMOKGILI(I)
580     PRINT I;" : ";TITLE$(I);TAB(30);HANGMOKGILI(I): IF
        HANGMOKGILI(I) > 48 THEN BEEP
590 NEXT
592 IF HANGMOKTTL > R.GILI THEN PLAY"cde"
600 PRINT "                               〈리턴〉 키를 누르세요. "
610 INPUT " ==================================================
=================",AA$
620 PRINT:PRINT" .... 정말로 처음입니까 ?  [ 1= 네, 0= 아니오 ] ";
630 INPUT ":",IBOK$ :IF IBOK$ = "1" THEN 700 ELSE  990 :REM
700 REM ******* 카드 화일의 구조 기록 *****************
710 PRINT" 화일의 구조를 디스켓에 기록중입니다......... "
712 INFILENAME$= "lkcard" + ".sif"
720 OPEN INFILENAME$ FOR OUTPUT AS 1
722     PRINT #1, R.GILI
724     PRINT #1, R.HANGMOKSOO
730 FOR J= 1 TO  8
740     PRINT #1, TITLE$(J)
750 NEXT
830 FOR J= 1 TO  8
840     PRINT #1, ENTRY$(J)
850 NEXT
900     REM  CLOSE #1
990 RETURN
1000 '========처음의 데이터 입력 ==================================
1010 '
1050 GOSUB 3000
1990 RETURN
1999 END
3000 '----- 입력 루틴 ----------------------------------------------
3010 CLS:KEY OFF
```

```
3012 LOCATE 25,1:PRINT"                                    ¦ 입력시
     수정키 [↑] = 바로 앞 항목으로 "
3020 LOCATE 1,1: PRINT "                              입 력  루 틴      ":
3022 LOCATE 1,67:PRINT"현재#";INT(LOF(1)/R.GILI);"까지"
3024 LOCATE 1,1:INPUT "입력할 자료 번호 (0= 신규 입력, 아니면 2-1000,

     -1=끝) ⊟  :",ENTRYNUMBER
3026  IF ENTRYNUMBER > 1000  THEN  PRINT CHR$(7) : GOTO  3010
3030 IF ENTRYNUMBER=0 THEN ENTRY0= 0:ENTRYNUMBER=INT(LOF(1)/
     (R.GILI)+1) ELSE ENTRYNUMBER=ENTRYNUMBER+1
3032 LOCATE 23,60:PRINT" # ";ENTRYNUMBER
3040 ENTRYNUMBER2= ENTRYNUMBER
3041 INPUTLINE=2+R.HANGMOKSOO
3042 LOCATE 2,3:PRINT LINE1$              '맨 윗줄 그리기
3043 FOR LL=3 TO INPUTLINE:LOCATE LL,3:PRINT "¦ ";         :
     LOCATE ,8:PRINT TITLE$(LL-2):LOCATE LL,28:PRINT"¦":LOCATE
     LL,77:PRINT" ¦":NEXT LL               '입력양식 테두리 그리기
3045 LOCATE 3+ R.HANGMOKSOO,3:PRINT "#=";:FOR I = 1 TO 36:PRINT
     "==";:NEXT I: PRINT "=#"
3060 '----- 엔트리 입력 루틴 ---------------------------------------
3067 FOR L=1 TO R.HANGMOKSOO:ENTRY$(L) = "":NEXT
3070 L=1
3071 '  FOR L=1 TO 8
3072 '      LOCATE 2+L, 29
3073 '      INPUT " ",ENTRY$(L)
3074 '  NEXT: GOTO 3195
3075 WHILE L > 0 AND L =< R.HANGMOKSOO
3080     V = 2 + L:LOCATE V,4:PRINT USING "###";L;
3082     ENTRYLEN=H(L)
3085     GOSUB 3300
3086 '
3090     IF (EOI=1) AND (L=1) THEN PRINT CHR$(7):GOTO 3115
3100     IF EOI=1 THEN L=L-1
3105     IF EOI=2 OR EOI=3 THEN L=L+1
3110     IF EOI=4 THEN L=STRINGFIELDS+1
3115 WEND
3116 '
3180 IF EOI=1 THEN L=STRINGFIELDS:GOTO 3075
3185 IF EOI=2 THEN 3070
3190 '--- 리턴으로 계속 내려오거나 CTRL-W가 눌려지면 데이터 기록 -------
3195 FOR L=1 TO 8
3196 '
3200 LSET A$(L)=ENTRY$(L)
3205 IF L=1 THEN LSET KA$(1)=ENTRY$(1) : IF ENTRY0 = 0 THEN
     ENTRYNUMBER2=INT(LOF(2)/(24)+1) ELSE ENTRYNUMBER2 = EN
     TRYNUMBER2+1
3207 IF L=1 THEN LSET KA$(2)=MKS$(ENTRYNUMBER2)
3210 NEXT L
3260 '
3265 LOCATE 3+ R.HANGMOKSOO,3:PRINT"===========================
=========================";
3268 '
3270 XX=FRE(0):PRINT CHR$(214)
3273 '--- 화일을 기록하기 위해 버퍼로 옮김 ----------------------------
3274 PUT #1,ENTRYNUMBER:PUT #2,ENTRYNUMBER2
3275 LOCATE 24,1:PRINT" 더 입력하시렵니까:( 7=계속 입력, '리턴'= 끝 )
          ";:DUIB$=INPUT$(1):IF DUIB$="7" THEN 3010
3280 RETURN
3290 PRINT" end"
3300 '---- 입력 스트링 ------------------------------------------------
3350 P=LEN(ENTRY$(L)):EOI=0
3355 '
3360 LOCATE V,30:PRINT ENTRY$(L);LEFT$(UNDERLINE$,ENTRYLEN-
     LEN(ENTRY$(L))):PRINT CHR$(213)
3365 LOCATE V,29+(P+1)
3370 CH$=INKEY$:IF LEN(CH$)=0 THEN 3370
3375 ON LEN(CH$) GOSUB 3400, 3500
3380 IF EOI THEN LOCATE V,30:PRINT ENTRY$(L);SPC(ENTRYLEN- LEN

  (ENTRY$ (L))+5);:LOCATE V,77:PRINT" ¦":RETURN ELSE GOTO 3365
3400 '
3410 IF CH$< " " OR CH$=CHR$(127) OR CH$=CHR$(255) THEN 3480
3420 IF P>= ENTRYLEN THEN BEEP:RETURN
3440 IF LEN(ENTRY$(L))=ENTRYLEN THEN ENTRY$(L)=LEFT$(ENTRY$(L),
     ENTRYLEN-1)
3450 ENTRY$(L)=LEFT$(ENTRY$(L),P)+CH$+MID$(ENTRY$(L),P+1)
3460 P=P+1:PRINT MID$(ENTRY$(L),P)
3470 RETURN
3480 '
3485 IF INSTR(CHR$(8)+CHR$(13)+CHR$(23)+CHR$(25)+CHR$(127),CH$)
     =0 THEN BEEP:RETURN
3490 IF CH$=CHR$(13) OR CH$=CHR$(23) THEN EOI=2+INT(ASC(CH$)/10)
3492 IF CH$=CHR$(25) THEN PRINT LEFT$(UNDERLINE$,ENTRYLEN-P):EN
     TRY$(L)=LEFT$(ENTRY$(L),P)
3495 IF CH$=CHR$(8) OR CH$=CHR$(127) THEN IF P>0 THEN ENTRY$(L)
     =LEFT$(ENTRY$(L),P-1)+MID$(ENTRY$(L),P+1):LOCATE V,29+P :
     PRINT MID$(ENTRY$(L),P);"_":P=P-1 ELSE BEEP
3497 RETURN
3500 '--- 방향 키 ------------------------------------------------
3510 IF ASC(CH$)<>0 THEN BEEP:RETURN
3520 DIRECTION=ASC(MID$(CH$,2))
3530 IF DIRECTION=72 THEN EOI=1           '윗방향 화살표
3540 IF DIRECTION=80 THEN EOI=2           '밑방향 화살표
3550 IF DIRECTION=75 THEN IF P>0 THEN P=P-1 ELSE BEEP
3560 IF DIRECTION=77 THEN IF P<LEN(ENTRY$(L)) THEN P=P+1 ELSE
     BEEP
3570 IF DIRECTION=71 THEN P=0
3580 IF DIRECTION=79 THEN P=LEN(ENTRY$(L))
3590 IF DIRECTION=83 THEN ENTRY$(L)=LEFT$(ENTRY$(L),P)+MID$(EN
     TRY$(L),P+2):PRINT MID$(ENTRY$(L),P+1);"_"
3600 RETURN
4100 '------자료 수정, 삭제하기 ---------------------------------------
4101 '
4102 IF SAKJE$="sakje" THEN CLS:LOCATE 3,20:COLOR 0,7: PRINT"
```

```
    자료 삭제하기 ":COLOR 7,0:PRINT:INPUT" 삭제할 번호는 # [1
    - 1000]: ⏎ ",DISPLAYENTRY:GOSUB 4840:RETURN
4104 IF SOOJUNG$="soojung" OR SOOJUNG$="finish" THEN CLS:LOCATE
    3,20:COLOR 0,7:PRINT" ■■■ 자료 수정하기 ■■■":COLOR 7
    ,0:PRINT
4105 IF SOOJUNG$="soojung" OR SOOJUNG$="finish" THEN INPUT" 수정
    하실 자료 번호는 [1 - 1000], 모르면 0 ⏎",DISPLAYENTRY : IF
    DISPLAYENTRY=0 THEN GOSUB 6210:RETURN ELSE GOSUB 4840
4106 IF SOOJUNG$="finish" THEN PRINT" 또 수정하십니까 (1= 또 수정,
    0= 끝)";:DISPLAYENTRY$=INPUT$(1):DISPLAYENTRY=VAL(DISPLAYEN
    TRY$):IF DISPLAYENTRY=1 THEN 4104 ELSE RETURN
4110 '
4120 RETURN
4140 '--- 편집 루틴 ---------------------------------------------
4142 PRINT
4144 CLOSE
4150 END
4180 RETURN
4200 REM ********** data search & data update ******************
4210 CLS:CLOSE
4220 REM
4230  FILENAME$ = "lkcard.sok"
4250 PRINT"... 잠간만 ....":PRINT
4260 '------- ....SOK 화일의 .....SIF 화일에서 정보를 얻는다.
4270 INFFILE$= "lkcard" + ".sif"
4271 OPEN INFFILE$  FOR INPUT  AS 1
4272  IF EOF(1) THEN 4360:REM close
4280    INPUT #1, R.GILI
4290    INPUT #1, R.HANGMOKSOO
4299    REM
4300  FOR J= 1 TO  8
4310      INPUT #1, TITLE$(J)
4320  NEXT
4330  FOR J= 1 TO  8
4332      INPUT #1, ENTRY$(J): H(J)= VAL(ENTRY$(J))
4334  NEXT
4340 HANGMOKGILI(1)=20:HANGMOKGILI(2)=4:HANGMOKGILI(3)=10
4342 HANGMOKGILI(4)=4:HANGMOKGILI(5)=12:HANGMOKGILI(6)=25
4344 HANGMOKGILI(7)=20:HANGMOKGILI(8)=16
4350 FOR II=1 TO 8
4354     H(II)=HANGMOKGILI(II)
4356 NEXT
4360 CLOSE #1
4370 OPEN "r",#1,FILENAME$,111
4380 FIELD#1,20 AS A$(1),4 AS A$(2),10 AS A$(3),4 AS A$(4),12
    AS A$(5),25 AS A$(6),20 AS A$(7),16 AS A$(8)
4390 KEYNAME$="lkcard"+".SKY"
4392 OPEN "r",#2,KEYNAME$,24
4393    FIELD #2, 20 AS KA$(1), 4 AS KA$(2)
4394    FOR R= 1 TO 1000
4395        GET #2,R
4396        KEYNAME$(R)= KA$(1): KEYRECORD(R)= CVS(KA$(2))
4397        IF R >= LOF(2)/24 THEN 4399
4398    NEXT:PRINT
4399 RETURN
4500 '--- 한 레코드 보여주기 루틴 ---------------------------------
4510 CLS
4520 PRINT:PRINT
4530 GET #1, DISPLAYENTRY
4534     IF LEFT$(A$(1),6)= "삭제됨" THEN 4661
4550 '
4552     IF REPLY=77 THEN LPRINT TAB(5);"# ";DISPLAYENTRY
4554 PRINT"════════════════════════════════════════"
4560 FOR ITEMNUMBER=1 TO 8
4570    ENTRY$(ITEMNUMBER)= A$(ITEMNUMBER)
4571    PRINT TAB(3);USING"##";ITEMNUMBER;
4580    PRINT TAB(6);TITLE$(ITEMNUMBER);TAB(30);":";ENTRY$(IT
    EMNUMBER)
4586    IF REPLY=7 THEN LPRINT TITLE$(ITEMNUMBER);" : ";ENTRY$(
    ITEMNUMBER),
4588    IF REPLY=77 THEN LPRINT " . ";ENTRY$(ITEMNUMBER),
4590 NEXT
4600 PRINT"────────────────────────────────────────"
4661 IF SAKJE$="sakje" OR SOOJUNG$="soojung" THEN IF LEFT$(A$(1
    ),6)="삭제됨" THEN PRINT"삭제된 데이타입니다.":PRINT
4662 IF SAKJE$="sakje" THEN SAKJE$="finish":INPUT "..... 삭제할
    자료가 맞습니까(1= 삭제, 0= 안 삭제):⏎ ",SAKJEOK$:IF SAKJEOK$<>
    "1" THEN RETURN ELSE LSET A$(1)="삭제됨"
    :PUT #1,DISPLAYENTRY:RETURN
4664 IF SOOJUNG$="soojung" OR SOOJUNG$="finish" THEN SOOJUNG$="
    finish":INPUT "...수정하실 자료입니까 (1= 수정, 0= 안 수정):⏎
    ",SOOJUNGOK$ :IF SOOJUNGOK$<>"1" THEN RETURN ELSE GOSUB
    6000:RETURN
4665 PRINT TAB(40);"# ";DISPLAYENTRY
4668 IF REPLY=77  OR REPLY= 7 THEN LPRINT " ..*"
4670 LOCATE 1,1
4680 RETURN
4682 '
4800 '--- 한 레코드 찾기 ------------------------------------------
4810 CLS:PRINT
4820 INPUT" 살펴볼 레코드 번호는  ( 1 - 1000 ) : ⏎ ",DISPLAYENTRY
4840    REM DISPLAYENTRY= DISPLAYENTRY + 1
4850 IF DISPLAYENTRY>LOF(1)/R.GILI THEN PRINT" 자료가 없읍니다. ":
    FOR PA=1 TO 1000:NEXT PA:GOTO 4890
4860 GOSUB 4500 :REM display data
4862 IF SOOJUNG$= "soojung" THEN 4890
4870 INPUT " ... 리턴을 누르면  계속됩니다. ",REPLY$
4880  IF REPLY$="" THEN 5000 ELSE 4670
4890 RETURN
4900 '--- 계속 찾기 -----------------------------------------------
4910 CLS:PRINT
4930 INPUT" 살펴볼 자료의 처음 번호는  : ⏎ ",DISPLAYENTRY
```

```
4940    REM
4950    IF DISPLAYENTRY > LOF(1)/R.GILI THEN 4990
4960    GOSUB 4500:REM  한 레코드 보여주기
4970 LOCATE 1,1:PRINT"다음목록을 보시려면 0, 차림표로 가려면 1을 누르세요.
    ";:REPLY$=INPUT$(1):REPLY=VAL(REPLY$)
4980 IF REPLY=1 THEN 4990
4982 DISPLAYENTRY=DISPLAYENTRY+1
4984 GOTO 4950
4990 RETURN
5000 '---- 데이터 찾기 루틴 --------------------------------------
5010 CLS
5012 SOOJUNG$="find" :SAKJE$="find":REM after soojung finish
5020 COLOR 0,7:LOCATE 3,7:PRINT"███      명함 (전화 번호) 찾기
    ███":COLOR 7,0
5030 LOCATE 5,5:PRINT"1. 레코드 번호로 찾기"
5040 LOCATE 6,5:PRINT"2. 연속해서 보여주기"
5050 LOCATE 7,5:PRINT"3. 항목별로 찾아보기 ( =, <, > ) "
5060 LOCATE 8,5:PRINT"4. 키(1번) 항목으로 찾기"
5070 LOCATE 10,5:PRINT"0. 시작 차림표로 돌아가기."
5080 LOCATE 21,10:PRINT" ... 어느 번호 :";:REPLY$=INPUT$(1):
    REPLY= VAL(REPLY$)
5082 IF REPLY$="0" THEN 20: REM  to menu
5084 ON REPLY GOSUB 4800,4900,5100,6200:REM 6200=key로 찾기
5090 RETURN :REM to menu
5100 '항목별로 찾기 ----------------------------------------------
5110 CLS :PRINT:PRINT
5111 LOCATE 7,1:FOR J=1 TO 3:LOCATE 7,(J*25-22):PRINT J;"]";TIT
LE$(J);:NEXT:PRINT
5112 LOCATE 8,1:FOR J=4 TO 6:LOCATE 8,(J*25-97):PRINT J;"]";TIT
LE$(J);:NEXT:PRINT
5113 LOCATE 9,1:FOR J=7 TO 8:LOCATE 9,(J*25-172):PRINT J"]";TIT
LE$(J);:NEXT:PRINT
5117 IF R.HANGMOKSOO<10 THEN LOCATE 10,1:PRINT SPACE$(80):LOCA
TE 11,1:PRINT SPACE$(80):LOCATE 12,1:PRINT SPACE$(80)
5118 IF R.HANGMOKSOO<13 THEN LOCATE 11,1:PRINT SPACE$(80):LOCATE
12,1:PRINT SPACE$(80)
5119 LOCATE 13,1:PRINT"-------------------------------------------
--------------------"
5120 INPUT"찾아볼 항목 번호는 [1 - 8] ⏎  : ",ITM
5122 PRINT" 찾는 내용의 첫자만 알 때는 +를 누르세요. 아니면 그냥 ⏎을 ";
5123 INPUT" ",CHUT$
5124 IF CHUT$="+" THEN PRINT"찾을 내용의 첫자는 ==> ";: GOTO 5140
5130 PRINT ITM;" 항목에서 찾을 데이타는 ==>  ";
5140      PRINT CHR$(213);:INPUT"",MATCHSTRING$
5150    REM
5151 IF CHUT$="+" THEN 5260:REM 첫자만 알 때 찾아보기
5152 PRINT"데이타와 같은것 = ,  보다 작은것 < ,  입력한 데이타보다 큰것을
    찾을때 > ."
5154 INPUT " =, <, >   중 어느 것 : ",BIGYO$
5160 L=LEN(MATCHSTRING$):
5162 '
5170 MATCHSTRING$=MATCHSTRING$+SPACE$(H(ITM) -L)
5180 LNGTH=LOF(1)/R.GILI
5182 IF BIGYO$ <>"=" THEN  5500 ELSE 5190
5190 FOR J=1 TO LNGTH
5200     GET #1,J
5202     IF LEFT$(A$(1),6)="삭제됨" THEN 5240
5210     A$=A$(ITM)
5220     IF A$=MATCHSTRING$ THEN GOSUB 5300
5230 '
5240 NEXT J
5250 RETURN
5260 '--- 첫 문자로만 찾기 ----------------------------------------
5261 LNGTH=LOF(1)/R.GILI
5262 FOR J=1 TO LNGTH
5264    GET #1,J
5266    IF LEFT$(A$(1),6)= "삭제됨" THEN  5290
5270    A$=A$(ITM)
5272    IF LEFT$(A$,LEN(MATCHSTRING$))= MATCHSTRING$ THEN GOSUB
        5300
5274 '
5290 NEXT
5298 RETURN
5300 '
5310 CLS
5320 LOCATE 1,1
5322 IF REPLY= 11 OR REPLY=77 THEN DISPLAYENTRY=J:GOTO 5370
5330 PRINT"찾았읍니다. # ";J   :PLAY "C"
5340 INPUT"보시겠읍니까(1= 봄, 11= 연속봄, 0= 안봄, 7=프린터로, 77=프린터로
연속봄)⏎",REPLY
5350 IF REPLY<>0 THEN 5360 ELSE 5400
5360 DISPLAYENTRY=J
5362 IF REPLY=77 THEN LPRINT CHR$(10):LPRINT CHR$(10):LPRINT"====
==========================================="
5370 GOSUB 4500:REM DISPLAY DATA
5372 IF REPLY= 11 OR REPLY= 77 THEN 5382
5380 PRINT"...........'리턴'을 누르면 계속 ,( 0= 그만보고 차림표로 )
    ";:GYESOK$= INPUT$(1)
5382 IF DISPLAYENTRY=>LOF(1)/R.GILI THEN 40
5384 IF GYESOK$="0" THEN 40
5390 REM IF GYESOK$=" " THEN 5400  ELSE 5382
5400 RETURN
5499 '--- 비교하기 루틴 -------------------------------------------
5500 FOR J= 1 TO LNGTH
5510   GET #1,J
5512   IF A$(1)="삭제됨" THEN 5680
5520   A$=A$(ITM)
5530   IF BIGYO$="<" THEN IF A$<=MATCHSTRING$ THEN GOSUB 5700
5532 '
5540   IF BIGYO$=">" THEN IF A$ >=MATCHSTRING$ THEN GOSUB 5700
5542 '
5680 NEXT J
```

```
5682 IF REPLY=77 THEN LPRINT"-------------------------------------
------------------":LPRINT CHR$(10):LPRINT CHR$(10)
5690 LOCATE 23,5:PLAY"CEG":INPUT"아무키나 누르시면 차림표로 갑니다. ",
     SIJAK$: GOTO 40
5700 CLS:LOCATE 1,1
5702 IF REPLY=11 OR REPLY=77 THEN 5710
5704 PLAY"D":PRINT"찾았읍니다. #";J
5706 INPUT"보시겠읍니까 (1= 봄, 11= 연속봄, 0= 안봄, 7=프린터로, 77=프린터
로 연속봄@ )";REPLY
5708 IF REPLY<>0 THEN 5710 ELSE 5750
5710 DISPLAYENTRY=J:GOSUB 4500
5712 IF REPLY=11 OR REPLY=77 THEN 5740
5720 PRINT"..........'리턴'을 누르면 계속 ,(0= 그만보고 차림표로 ) ";:
     GYESOK$=INPUT$(1)
5730 IF GYESOK$="0" THEN 40
5740 IF DISPLYENTRY>=LOF(1)/R.GILI THEN 40
5750 RETURN
6000 '--- 수정하는 루틴 -------------------------------------------
6010 PRINT"-----------------------------------------------------"
6014 '
6020 '
6022 LOCATE 21,1:INPUT"  수정하실 항목 번호는 # ",ITEMNUMBER
6024 IF ITEMNUMBER<1 OR ITEMNUMBER>R.HANGMOKSOO THEN BEEP :GOTO
     6020
6030 PRINT"(한글 모드임.)";CHR$(213);:INPUT" ",SOOJUNGSTRING$:L
SET A$(ITEMNUMBER)=SOOJUNGSTRING$:LOCATE ITEMNUMBER+3,31:PRINT
SPACE$(48):LOCATE ITEMNUMBER+3,31:COLOR 0,7:PRINT SOOJUNGSTRING$
6040 COLOR 7,0
6050 LOCATE 22,1:PRINT "또 수정할 항목이 있읍니까 <Y/N> ";:SOOJUNGAGA
IN$=INPUT$(1)
6060 IF SOOJUNGAGAIN$="Y" OR SOOJUNGAGAIN$="y"
     THEN LOCATE 21,1:PRINT  SPACE$(70):LOCATE 22,1:PRINT
     SPACE$(70):LOCATE 23,1:PRINT SPACE$ (70):GOTO 6020
6080 PUT #1,DISPLAYENTRY
6082 PRINT
6084 '
6100 RETURN
6200 '--- 키 항목으로 찾기 ----------------------------------------
6210 CLS:PRINT TAB(15);"키 항목으로 찾아 보기"
6220 LOCATE 3,1:PRINT TITLE$(1);
6230 PRINT"항목에서 찾을 데이타는 ==>";CHR$(213);:INPUT MATCHSTRING$
6231 L=LEN(MATCHSTRING$)
6232 IF L>=20 THEN MATCHSTRING$=LEFT$(MATCHSTRING$,20):GOTO 6240
     ELSE GOTO 6233
6233 MATCHSTRING$=MATCHSTRING$+SPACE$(20-L)
6240 FOR J=1 TO R
6250  IF KEYNAME$(J)= MATCHSTRING$ THEN PLAY"C" :DISPLAYENTRY=
      KEYRECORD(J):GOSUB 4500:GOSUB 6300
6260 NEXT J
6270 '
6290 RETURN
6300 PRINT"같은 이름의 데이타가 더 있나요 (1= 또 있음, 0= 없음)";:REPLY$=
     INPUT$(1):REPLY=VAL(REPLY$)
6310 IF REPLY=1 AND J<R THEN RETURN ELSE 40
10000 '
10002 CLS
10010 LOCATE 3,20:PRINT"&&&  도 움 말  &&&"
10020 PRINT"..............................................."
10030 PRINT"1. 한 레코드의 길이는 110 바이트 (한글 55자까지)입니다."
10102 PRINT"2. lkcard.sif 화일에  레코드 길이, 항목수,  항목의 길이가
      들어있읍니다."
10110 PRINT"3. 키 항목은  이름 항목입니다. "
10980 PRINT
10990 PRINT"..............................................."
11000 INPUT" '리턴'을 누르시면 차림표로 갑니다.",HELP$
11010 IF HELP$="" THEN 11020 ELSE 11000
11020 RETURN
20000 '하드 카피
20010 '
30000 '
30002 CLOSE
30010 CLS:FILES:PRINT:PRINT
30011      KEYFILENAME$ = "lkcard.sky"
30012 REM
30014 IF RIGHT$(KEYFILENAME$,3)="sky" OR RIGHT$(KEYFILENAME$
      ,3)="SKY" THEN 30020 ELSE GOTO 30011
30020 OPEN "R",#1,KEYFILENAME$,24
30030 FIELD #1, 20 AS KA$(1),  4 AS KA$(2)
30040 GET #1,1
30050   KA=CVS(KA$(2))
30060 PRINT 1,KA$(1),KA
30061 IF KA=1 THEN 30062 ELSE BEEP:PRINT"명함관리의 키화일이 아닙니다."
      :PRINT:FOR PA=1 TO 1000:NEXT PA
30062 PRINT"--------------------------------------------------"
30070 FOR I= 1 TO 1000
30080 IF I>LOF(1)/24 THEN PLAY"c":PRINT:PRINT" *** meet eof
      *** ":FOR PA=1 TO 1000:NEXT:PRINT:GOTO 30130
30090     GET #1,I
30100     KEYNAME$   = KA$(1):  KEYRECORD   = CVS(KA$(2))
30102 IF I=1 THEN PRINT 1;") ";"#1 record ":PRINT KEYNAME$;"
      = # ";KEYRECORD:GOTO 30120
30110     PRINT "[";I;"]  ";KEYNAME$ ;" = # ";KEYRECORD  ;" ";
30120 NEXT
30130 PRINT
30140 CLOSE
30150 INPUT" ⏎을 누르시면 차림표로 갑니다. ",CHA$
30200 RETURN
30300 END
30333 '
60000 PLAY "b0c0b0c0"
60010 PRINT:PRINT
60100 CLOSE
```

```
60120 PRINT" Error Code= ";ERR
60130 PRINT" Error Line= ";ERL
60200 PRINT:COLOR ,7:INPUT"아무 키나 누르면 시작 차림표로 갑니다.",AMO
OKEY$
60210 COLOR 7,0:
61000 RESUME 15
```

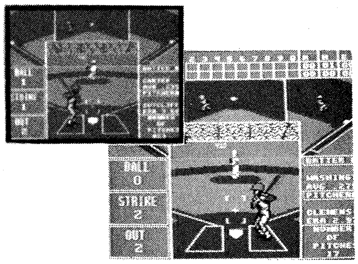
〔리스트 2〕 LKCARDPR.BAS

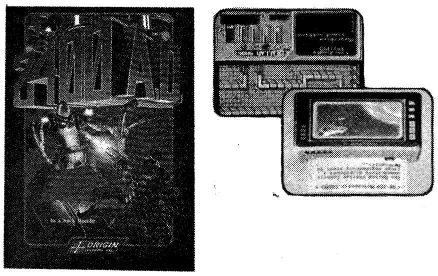
```
2 '------- Created by Lee Kisung -----------------------------
10 'SAVE "c:lkcardpr"
12 '------ on error rtn added
13 '
14 DIM A$(20) ,TITLE$(20), ENTRY$(20), ENTRY#(20),ITEMSOO(20),
      ITEMTOTAL#(20),K(60):DIM KA$(20)
15 DIM B$(20),NEWTITLE$(20),OLDTITLE$(20),NEWITEM(20),OLDITEM
(20),NEWITEM$(20):DIM OLDITEM$(20)
16 DIM HANGMOKGILI(20),H(20)
17 ON ERROR GOTO 60000
18 GOSUB 4200
19 '화일을 연다.
20 GOSUB 100 :GOSUB 120
21 FOR I3=1 TO 20:ITEMSOO(I3)=0:ITEMTOTAL#(I3)=0:NEXT I3
22    FOR I=1 TO 60:K(I)=0:NEXT I
28 '
30 LOCATE 19,3:PRINT"                    어느 번호를 원하십니까 (1--6)
   : ",: GOGO$=INPUT$(1)
34 GOGO=VAL(GOGO$):ON GOGO  GOSUB 20000 ,300,300,300,300,4140
40 GOTO 20
50 END
100 CLS
102 LOCATE 3,14:COLOR 0,7:PRINT"***█    명함(전화 번호) 인쇄  차림표
   █***":COLOR 7,0
103  PRINT:PRINT" 만듬 0.8 ";
104  PRINT TAB(60)"장왕사 전산실"
105  PRINT:PRINT"#==========================================#"
110  LOCATE 8,3:PRINT CHR$(212)+CHR$(218);:FORI=1 TO 30:PRINT
     CHR$(212)+CHR$(196);:NEXT I:PRINT CHR$(212)+CHR$(191)
112 FOR LJ=9 TO 17:LOCATE LJ,3:PRINT CHR$(212)+CHR$(179):LOCATE
    LJ,65:PRINT CHR$(212)+CHR$(179):NEXT LJ
114 LOCATE 17,3:PRINT CHR$(212)+CHR$(192);:FOR I=1 TO 30: PRINT
     CHR$(212)+CHR$(196);:NEXT I:PRINT CHR$(212)+CHR$(217)
118 RETURN
120 '
127 LOCATE  9,9:PRINT      "1. 명함(전화 번호) 목록 "
130 LOCATE 12,9: PRINT     "3. 디스켓 화일 목록 살피기 "
136 LOCATE 15,9: PRINT    "6. 끝  (시작 프로그램으로)"
200 RETURN
300 CLS:FILES:PRINT:INPUT" .. 리턴을 누르시면 차림표로 갑니다.",CHA$
310 RETURN
1000 '
2000 '
4140 CLOSE                          '작업을 끝냄
4153 PRINT
4154 PRINT" 감사합니다. "
4156 RUN"lkcard"
4160 END
4190 RETURN
4200 '--- 화일이름을 입력하고 오픈한다  ------------------------------
4210 CLS:CLOSE
4230 PRINT
4242       FILENAME$  = "lkcard.sok"
4260 '_______.SOK 화일의 _______.SIF 화일에서 정보를 읽는다.
4270 INFFILE$= "lkcard" +".sif"
4271 OPEN INFFILE$  FOR INPUT  AS 1
4272 IF EOF(1) THEN 4360
4280 INPUT #1, R.GILI
4290 INPUT #1, R.HANGMOKSOO
4299 '
4300 FOR J=1 TO 8
4310    INPUT #1,TITLE$(J)
4320 NEXT
4330 FOR J=1 TO 8
4340    INPUT #1,ENTRY$(J):H(J)=VAL(ENTRY$(J))
4350 NEXT
4352 H(1)=20:H(2)=4:H(3)=10
4354 H(4)=4:H(5)=12:H(6)=25
4356 H(7)=20:H(8)=16
4360 CLOSE #1
4370 OPEN "r",#1,FILENAME$,111
4380 FIELD#1,H(1)AS A$(1),H(2)AS A$(2),H(3)AS A$(3),H(4)AS A$(4
),H(5)AS A$(5),H(6)AS A$(6),H(7)AS A$(7),H(8)AS A$(8)
4398 RETURN
4399 '--- 화일을 오픈하고 읽는다 -----------------------------------
4400 '
4450 RETURN
20000 '하드 카피
20010 '
20150 '
20200 GOTO 20460
20240 '--- 한 레코드 내용 ----------------------------------------
20250 CLS
20260 IF DISPLAYENTRY>LOF(1)/111 THEN GOTO 20450
20270 GET #1,DISPLAYENTRY
20280 FOR ITEMNUMBER=1 TO R.HANGMOKSOO
20290    ENTRY$(ITEMNUMBER)=A$(ITEMNUMBER)
20300 NEXT ITEMNUMBER
```

```
20340 FOR IREPORT=1 TO 8
20360     IF IREPORT=8 AND HWOISA$<>"1" THEN 20430
20362     IF PRINTER$="ON" THEN LPRINT ENTRY$(IREPORT);
20370     PRINT ENTRY$(IREPORT);
20430 NEXT IREPORT
20440 PRINT
20442 IF PRINTER$="ON" THEN LPRINT:LPRINT"......................
      ..............................."
20450 RETURN
20459 '
20460 CLS
20461 RECCOUNT=0
20462 FOR I=1 TO 60:K(I)=0:NEXT I
20470 FOR JJ=1 TO 3:LOCATE 10,(JJ*25-22):PRINT JJ;")";TITLE$(JJ)
      ;:NEXT JJ:PRINT
20472 FOR JJ=4 TO 6:LOCATE 11,(JJ*25-97):PRINT JJ;")";TITLE$(JJ)
      ;:NEXT JJ:PRINT
20474 FOR JJ=7 TO 8:LOCATE 12,(JJ*25-172):PRINT JJ;")";TITLE$(JJ)
      ;:NEXT JJ:PRINT
20490 PRINT"══════════════════════════════":PRINT
20500 HR$="장 왕 사 "
20766 '
20770 INPUT"* 찾는(인쇄할) 자료 선택은 몇번 항목(1--8), 전 자료 다 찍기= 0
      : ⏎ ",ITM$:ITM=VAL(ITM$)
20780 IF L$<>"Y" THEN L$=""
20782 IF ITM=-1 OR ITM=0 THEN 20830
20784 PRINT ITM;" = ";TITLE$(ITM)
20790 IF ITM<9 THEN 20792 ELSE BEEP:GOTO 20770
20792 PRINT" 찾는 내용의 첫자만 알 때는 '+'를 누르세요, 아니면 그냥  ⏎을
      ";
20794 INPUT "",CHUT$
20796 IF CHUT$="+" THEN PRINT"찾을 내용의 첫자는 ==> ";:GOTO 20802
      ELSE 20800
20800 PRINT"찾을 내용은 ";
20802 PRINT CHR$(213);:INPUT": ",MATCHSTRING$:GOTO 20812
20810 '
20812 IF CHUT$="+" THEN 20830
20820 INPUT"같은 것=, 작은 것<, 입력한 내용보다 큰 것을 찾을때 >⏎",BIGYO$
20830 INPUT"인쇄하실 목록의 제목은:⏎ ",BOGOSUNAME$
20840 COLOR 0,7:PRINT" 프린터를 준비하세요 ";:COLOR 7,0:PLAY "C":
20850 INPUT"  프린터에=1, 아니면=0 ⏎",SIJAK$
20852 INPUT" 8번 항목(비고)도  찍으십니까  (찍음=1, 안찍음=0 (보통 =0)
      ⏎ ",HWOISA$
20856 '--- 자료선택 완료 ---------------------------------------------
20860 PRINTER$="OFF":IF SIJAK$="1" THEN PRINTER$="ON":LPRINT " ":
      LPRINT CHR$(27)+"h"
20861 IF SIJAK$="1" THEN PLAY "C":INPUT" 프린터는 1=큐닉스(후지쓰),
      0=삼보(엡손)  ...... ⏎ .",QNIX$:IF QNIX$="1" THEN LPRINT
      CHR$(27)+"h"+CHR$(1)+CHR$(27);"g";CHR$(1)
20870 CLS:LOCATE 1,10:COLOR 0,7:PRINT BOGOSUNAME$:COLOR 7,0
20872 IF CHUT$="+" THEN 21600
20880 '하드 카피
20890 L=LEN(MATCHSTRING$)
20892 IF ITM=-1 OR ITM=0 THEN 20910
20900 MATCHSTRING$=MATCHSTRING$+SPACE$(H(ITM)-L)
20910 LNGTH=LOF(1)/R.GILI
20920 IF ITM=-1 OR ITM=0 THEN 21380
20930 IF BIGYO$<>"=" THEN 21130 ELSE 20940
20940 LINECOUNT=0
20941 IF HWOISA$="1" THEN JUL=2 ELSE JUL=1
20942 FOR J=1 TO LNGTH
20950     GET #1,J
20960     IF LEFT$(A$(1),6)="삭제됨" THEN 21000
20970     A$=A$(ITM)
20980     IF A$=MATCHSTRING$ THEN GOSUB 21080
20990 '
21000 NEXT J
21010 PRINT"-----------------------------------------------".
21020 IF PRINTER$="ON" THEN LPRINT:LPRINT"---------------------
      -----------------------"
21030 GOSUB 21300
21040 IF PRINTER$="ON" THEN LPRINT CHR$(10):LPRINT CHR$(10):LP
RINT CHR$(10)
21050 PLAY"CEG"
21060 INPUT" 아무 키나 누르시면 차림표로 갑니다.",CH$
21070 GOTO 20
21080 PRINT
21090 DISPLAYENTRY=J
21100 GOSUB 20240
21110 IF DISPLAYENTRY>=LOF(1)/R.GILI THEN 20
21112 LINECOUNT=LINECOUNT+JUL:IF LINECOUNT=>56 THEN GOSUB 21550
21120 RETURN
21130 '하드 카피
21140 GOSUB 21550
21150 FOR J=1 TO LNGTH
21160     GET #1,J
21170     IF LEFT$(A$(1),6)= "삭제됨" THEN 21230
21172     IF ITM=-1 OR ITM=0 THEN GOSUB 21250:GOTO 21230
21180     A$=A$(ITM)
21190    IF BIGYO$="<" THEN IF A$<=MATCHSTRING$ THEN GOSUB 21250
21200 '
21210    IF BIGYO$=">" THEN IF A$>=MATCHSTRING$ THEN GOSUB 21250
21220 '
21230 NEXT J
21240 GOTO 21010
21250 DISPLAYENTRY=J
21252 GOSUB 20260:GOTO 21260
21253 LINECOUNT=LINECOUNT+JUL:IF LINECOUNT=>56 THEN GOSUB 21550
21254 GOTO 21280
21260 LINECOUNT=LINECOUNT+1
21270 IF LINECOUNT=>60 THEN GOSUB 21550
21280 IF DISPLAYENTRY>=LOF(1)/R.GILI THEN 21240
```

```
21290 RETURN
21300 '--- 전부 프린트 ------------------------------------------
21310 IF ITM>0 THEN 21370
21320 IF PRINTER$="ON" THEN LPRINT RECCOUNT ELSE PRINT RECCOUNT
21370 PRINT:RETURN
21380 '
21390 GOSUB 21550
21450 LINECOUNT=0
21460 FOR J= 1 TO LNGTH
21470     GET #1,J
21480     IF LEFT$(A$(1),6)= "삭제됨" THEN  21530
21490     DISPLAYENTRY=J:GOSUB 20260:RECCOUNT=RECCOUNT+1
21500     IF HWOISA$="1" THEN LINECOUNT=LINECOUNT+2 ELSE
          LINECOUNT=LINECOUNT+1
21510     IF LINECOUNT>=60 THEN GOSUB 21550
21520     IF DISPLAYENTRY>=LOF(1)/111 THEN 21540
21530 NEXT J
21540 GOTO 21010
21550 IF PRINTER$="ON" THEN LPRINT CHR$(12):LPRINT TAB(20)
      BOGOSUNAME$;  TAB(60);DATE$:LPRINT " ================
==========================================="
21560 CLS:PRINT TAB(20) BOGOSUNAME$
21570 LINECOUNT=0
21580 RETURN
21590 CLOSE
21600 '
21602 LNGTH=LOF(1)/R.GILI
21604 FOR J=1 TO LNGTH
21606 GET #1,J
21608 IF LEFT$(A$(1),6)="삭제됨" THEN 21620
21610 A$=A$(ITM)
21612 IF LEFT$(A$,LEN(MATCHSTRING$))=MATCHSTRING$ THEN GOSUB
      21080
21620 NEXT
21628 PRINT"-----------------------------------------------"
21630 GOTO 21020
21640 '
23000 END
50000 '
50010 JARI=LEN(ENTRY$)
50020 WHILE JARI>0
50030   IF MID$(ENTRY$,JARI,1)>" " THEN ENTRY$=LEFT$(ENTRY$
        ,JARI):JARI=0
50040   JARI=JARI-1
50050 WEND
50100 RETURN
50200 '
60000 PLAY "b0c0b0c0"
60001 '
60010 PRINT:PRINT
60100 CLOSE
60120 PRINT" Error Code= ";ERR
60130 PRINT" Error Line= ";ERL
60200 PRINT:COLOR .7:INPUT"아무 키나 누르면 시작 차림표로 갑니다. ",
      AMOOKEY$
60210 COLOR 7,0
61000 RESUME  17
```







| | |
|---|---|
| 10TH FRAME PROBOWLING | 1 |
| 221B BAKER STREET | 2 |
| **2400 AD** | 1 |
| 3D HELICOPTER | 1 |
| 4TH & INCHES | 1 |
| ACE OF ACE | 1 |
| ALTERNATE REALITY | 2 |
| ANCIENT ART OF WAR AT SEA | 1 |
| BAD STREET BRAWER | 1 |
| BADUK | 1 |
| BALANCE OF POWER | 3 |
| BARD'S TALE I | 2 |
| **BARD'S TALE II** | 2 |
| BATTLE CHESS | 2 |
| BATTLE ZONE | 1 |
| BATTLEHAWKS 1942 | 2 |
| BIONIC COMMANDO | 1 |
| BLACK CAULDRON | 2 |
| BLACK JACK ACAEMY | 1 |
| BLOCK BUSTER | 1 |
| BOP,N WRESTLE | 1 |
| BORROWED TIME | 1 |
| BUBBLE GHOST | 1 |
| CALIFONIA GAME | 1 |
| CARD SHARKS | 1 |
| CHAMPIONSHIP BASEBALL | 1 |
| CHAMPIONSHIP BOXING | 1 |
| CHAMPIONSHIP GOLF | 1 |
| CHUCK YEAGER'S FLIGHT TRAINNER | 1 |
| COMMANDO | 1 |
| CONTRA | 1 |
| CRAZY CARS | 1 |
| CRIME & PUNISHMENT | 1 |
| DAM BUSTER | 1 |
| DARK CASTLE | 2 |
| DEATH SWORD | 1 |
| DEFENDER OF THE CROWN | 1 |
| DESTROYER | 1 |
| DOUBLE DRAGON | 2 |
| **DOWNHILL CHALLENGE** | 2 |
| DRAGON WORLD | 2 |
| EARL WEAVER BASE BALL | 2 |
| **ECHELON** | 2 |
| ELITE | 1 |
| EMPIRE | 1 |
| F-15 STRIKE EAGLS | 1 |
| F-19 STEALTH FIGHTER | 3 |
| FALCON F-16 | 1 |
| FIRE POWER | 2 |
| FLIGHT SIMULATOR | 1 |
| FLIGHT SIMULATOR V3.0 | 2 |
| FORD SIMULATOR | 1 |
| FRIENDLY WARE GAME | 1 |
| **FUTURE MAGIC** | 2 |
| GAMESTAR BASEBALL | 1 |
| GATO | 1 |
| GAUNTLET | 1 |
| GHOSTS'N GOBLINS | 1 |
| **GOLD RUSH** | 5 |
| GRAND PRIX CIRCUIT | 2 |

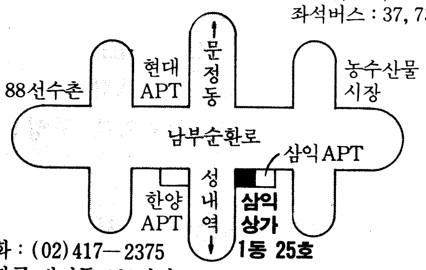
| | |
|---|---|
| GUNSHIP | 2 |
| HACKER II | 1 |
| HARD BALL | 1 |
| HARRIER | 2 |
| HEROES OF LANCE | 3 |
| HULK | 1 |
| HUNT FOR RED OCTOBER | 1 |
| IKARI WARRIOR | 1 |
| IMAGE TENNIS | 1 |
| IMPOSSIBLE MISSION III | 1 |
| INDOOR SPORTS | 2 |
| INFILTLATOR | 1 |
| INTO THE EAGLE'S NEST | 1 |
| JACK NIKLAUS GOLF | 2 |
| JEOPARDY | 1 |
| JEOPARDY 88 | 1 |
| JET | 1 |
| KARNOV | 2 |
| KING'S QUEST I | 1 |
| KING'S QUEST II | 2 |
| KING'S QUEST III | 3 |
| KING'S QUEST IV | 9 |
| KNIGHT ORC | 2 |
| LAST NINJA | 2 |
| LEISURE SUIT LARRY II | 6 |
| LOCK ON | 2 |
| LODE RUNNER | 1 |
| MAD SPY VS SPY II | 1 |
| MAN HUNTER | 5 |
| MANIAC MANSION | 1 |
| MARBLE MADNESS | 2 |
| MASTER NINJA | 1 |
| MIGHT & MAGIC | 2 |
| MINI PUTT | 1 |
| MIXED UP MOTHER GOOSE | 2 |
| MOEBIUS | 2 |
| NFL CHALLENGE | 2 |
| NINJA | 1 |
| OFF SHORE WARRIOR | 1 |
| ONE ON ONE | 1 |
| PATTON VS ROMMEL | 1 |
| PETE ROSE PENANT FEVER | 2 |
| PHANTASIE | 1 |
| PHANTASIE III | 1 |
| PHYCHO | 2 |
| PINBALL WIZARD | 1 |
| PIRATES | 2 |
| PLATOON | 2 |
| POLICE QUEST II | 3 |
| POLICE QUEST | 6 |
| POOL OF RADIANCE | 3 |
| PT-109 | 1 |
| QUSTRON II | 2 |
| RAD WARRIOR | 1 |
| RAMPAGE | 1 |
| RINGS OF ZILFIN | 2 |
| ROADWAR 2000 | 1 |
| **ROCK FORD** | 2 |
| **ROCKET RANGER** | 1 |
| ROMANCE OF THREE KINGDOM | 3 |

| | |
|---|---|
| **RUSH'N ATTACK** | 1 |
| SCRABBLE | 1 |
| SDI | 1 |
| SEARCH AND DISTROY | 1 |
| SEVEN CITY OF GOLD | 1 |
| SHADOW GATE | 2 |
| SHANGHAI | 1 |
| SHOGUN | 1 |
| SILENT SERVICE | 1 |
| SKATE OR DIE | 2 |
| SKY FOX II | 1 |
| SKY RUNNER | 1 |
| SPACE QUEST I | 2 |
| SPACE QUEST II | 3 |
| **SPACE QUEST III** | 6 |
| **SPORTING NEWS BASEBALL** | 1 |
| STAR COMMAND | 3 |
| STAR FLEET I | 1 |
| STAR FLIGHT | 2 |
| STAR GLIDER | 1 |
| STAR TREK | 1 |
| STREET FIGHTER | 2 |
| SUB BATTLE | 1 |
| SUMMER GAME II | 1 |
| SUPERBYKE CHALLENGE | 1 |
| SUPERSTAR ICEHOCKEY | 1 |
| TEST DRIVE | 1 |
| TETRIS | 1 |
| THE FELLOWSHIP OF THE RING | 1 |
| THE GANES SUMMER GAME EDITION | 4 |
| THE LAND OF THE LOUNGE LIZARDS | 2 |
| THEXDER | 2 |
| THREE STOOGES REEL | 2 |
| TKO | 1 |
| TOMAHAWK | 1 |
| TOP GUN | 1 |
| TREASURE ISLAND | 2 |
| TROJAN | 1 |
| TWILIGHT RANSOM | 2 |
| ULTIMA I | 1 |
| ULTIMA II | 3 |
| ULTIMA III | 2 |
| ULTIMA IV | 2 |
| ULTIMA V | 4 |
| VIDEO VEGAS | 1 |
| **WASTELAND** | 5 |
| WHERE IN THE EUROPE IS CARMEN- | 1 |
| WHERE IN THE USA IS CARMENSAN | 1 |
| WHERE IN THE WOLRD IS CARMEN | 1 |
| WHO FRAMED ROGER RABBIT | 2 |
| WILLOW | 2 |
| WINTER GAMES | 1 |
| WIZARD WARS | 2 |
| WIZARDRY I | 2 |
| WIZARDRY III | 1 |
| WIZARDRY IV | 1 |
| **WIZARDRY V** | 3 |
| WORLD GAME | 1 |
| WORLD GREATEST BASEBALL | 1 |
| WORLD KARATE CHAMPIONSHIP | 1 |
| WORLD TOUR GOLF | 1 |

**◎ 4월중순 발매된 SIERRA GAME 중 SPACE QUESTⅢ 와 GOLD RUSH를 긴급 입수 판매 합니다.**

## EYE아이콤

일반버스 : 33, 812, 137-1
좌석버스 : 37, 732




전화 : (02) 417—2375
송파구 방이동 398번지
가락 삼익상가 1층25호

## 회원모집 안내

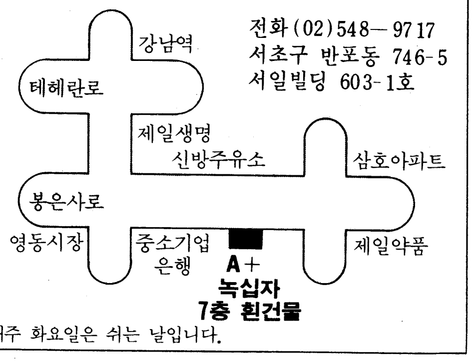
년가입비 : 10,000
월 비 : 10,000

## 지방 USER안내

1. 지방회원은 송료만 추가하시면 됩니다.
2. 필요한 SOFT WARE는 소액환으로 보내주시면 프로그램을 보내드립니다.

## A+ 에이플러스

전화 (02) 548—9717
서초구 반포동 746-5
서일빌딩 603-1호




매주 화요일은 쉬는 날입니다.

# 서브루틴의 사용

박승제/ 여주컴퓨터학원장

서브루틴(Subroutine)이란 스스로는 실행하지 않고 메인루틴(Mainroutine)의 제어하에 실행되는 일련의 명령어의 집합이다. 서브루틴을 적절히 활용하면 프로그램이 보기에도 좋을 뿐만 아니라 나중에 프로그램의 보수가 쉬워지는 등의 장점이 많다. 이번 호에는 서브루틴의 사용법에 대해서 설명한다.

## CALL과 RET

[형식]

```
       CALL 〈레이블명〉
          ∫
레이블:  ┌──────────┐
         │ 서브루틴  │
         └──────────┘
       RET
```

CALL명령은 레이블이 붙은 곳으로 실행위치를 옮기는 역할을 하며 서브루틴의 RET(Return) 명령에 의해 호출측으로 되돌아 온다. 서브루틴안에서 다른 서브루틴을 호출할 수도 있으며, 이 경우에 RET은 가장 가까운 CALL로 정확하게 되돌아 간다. 이것은 베이직의 GOSUB~RETURN과 아주 흡사하다.

어셈블리에서는 베이직과 마찬가지로 메인루틴과 서브루틴의 구별이 사실상은 없으므로 메인루틴이 끝난 후에 또다시 서브루틴이 수행되지 않도록 하는 장치가 반드시 필요하다. 이를테면 보기 1과 같은 베이직 프로그램을 보면 메인프로그램이 끝난 후에도 서브루틴을 수행하게 되므로 실행 후에는 'Return without gosub in 110' 에러가 발생하게 된다. 어셈블리에서는 이런 에러가 나오는 대신에 대부분 프로그램이 폭주해 버린다. 보기 1의 프로그램은 '50 END'를 기술하므로써 에러를 방지할 수 있다.

보기 2의 경우는 서브루틴을 메인루틴의 앞에 기술한 예인데 이 또한 에러를 유발한다. 보기 2는 'GOTO 100'이 누락되어 있다. 그러면 예제를 가지고 서브루틴의 사용법을 익혀보기로 하자.

〈예제 1〉

0H~FH까지 16진수를 화면에 출력한다.

〈처리 방법〉

0, 1, 2, 3…의 아스키코드가 48, 49, 50…이므로 0~9사이의 수치는 그 수에 48을 더하면 해당 문자의 아스키코드가 구해진다. 또 10, 11, 12, 13…은 'A', 'B', 'C'…에 대응하는데 A의 아스키코드는 65이므로 10~15의 수치는 55(65−10=55)를 더하면 해당하는 문자(A~F)의 아스키코드로 바뀌게 된다. 이것은 수치를 16진수로 바꿀 때 정석과 같이 이용되는 기법이므로 잘 익혀두어야 한다.

0~15의 수치는 CL을 이용해서 하나씩 증가시키기로 하고 수치를 문자로 바꾸는 것은 서브루틴에서 처리하도록 한다. 서브루틴에서는 CL의 내용을 DL로 옮긴 다음 DL의 값을 조사하여 DL이 0~9사이의 값이면 여기에 48을 더하고, 0~9의 값이 아니면 55를 더한 다음 DOS 기능호출 2번의 기능을 이용해서 화면에 1문자를 출력한 후 리턴하게 된다. 레지스터를 하나씩 증가시키는 것은 지난호에서 배운 'INC' 명령을 이용한다(리스트 1 참조).

## 스택의 이용

택시를 타면 주행거리 표시기(미터기)밑에 동전을 차곡차곡 쌓아두는 통을 보게 된다. 이 통은 위에서부터 동전을 넣게 되며 밑에 스프링이 달려있어서 꺼낼 때도 위에서 부터 꺼낸다.

〔보기 1〕 메인루틴에서 호출하지 않았는데도 서브루틴을 수행하는 예

```
10  A=10
20  B=20
30  GOSUB 100
40  PRINT C
100 C=A*B/2
110 RETURN
```

※ 50행에 END문이 누락되어 있다.

〔보기 2〕

```
10  C=A*B/2
20  RETURN
100 A=10
110 B=20
120 GOSUB 10
130 PRINT C
140 END
```

※ 5행에 GOTO 100이 누락되어 있다.

이 동전통은 입구와 출구가 같기 때문에 맨 나중에 넣는 동전이 맨 처음에 나오게 되고, 맨 먼저 넣은 동전은 맨 나중에야 나오게 된다.

컴퓨터에서는 이와같은 구조를 한 메모리를 스택(Stack)이라고 부르며 '맨 나중에 넣은 자료가 맨 처음에 나온다'는 개념을 특별히 LIFO(Last In First Out)라고 부른다. 이 스택은 메모리의 특정한 구역을 배정받는데 초기치는 DOS가 설정하는 것이지만 SS(스택 세그먼트)를 설정해주므로써 사용자가 스택 위치를 변경할 수도 있다.

이 스택에는 현재 몇개의 자료가 저장되어 있는지를 기억하는 레지스터가 있으며 이것을 스택 포인트(Stack Point ; SP 레지스터)라고 부른다. 스택에는 한꺼번에 2바이트를 단위로 저장과 제거를 하며, 자료를 저장하면 스택포인터가 2씩 증가를 하고, 제거를 할 경우에는 2씩 감소를 하게 된다. 스택에 데이터를 넣는 명령을 PUSH라 하며 스택에서 데이터를 꺼내는 명령을 POP이라고 부른다. 스택은 레지스터나 메모리의 내용을 잠시 대피시키거나 서브루틴을 호출할 경우에 자주 이용된다.

[형식]
PUSH 〈16비트 레지스터 / 16비트 메모리〉
POP 〈16비트 레지스터 / 16비트 메모리〉

위에서 16비트 레지스터란 범용 레지스터와 세그먼트 레지스터를 모두 포함하여 이르는 말이다.

스택에는 16비트의 자료는 모두 저장할 수 있지만 IP(Instruction Pointer) 레지스터는 PUSH, POP할 수 없으며 'POP CS' 명령은 존재하지 않는다. 또한, 플래그 레지스터를 스택에 대피 복구하는 명령은 'PUSHF'와 'POPF'가 별도로 마련되어 있다.

스택은 자료가 들어가는 순서와 나오는 순서가 역순이므로 이 점에 주의하여야 하며 자료를 넣는 갯수와 꺼내는 갯수가 일치하여야 한다.

[리스트 1] 0~FH를 화면에 표시한다.

```
0000                   MAIN      SEGMENT
                                 ASSUME  CS:MAIN
0000  B1 00                      MOV     CL,0
0002  E8 0010 R        START:    CALL    DISPLAY
0005  FE C1                      INC     CL
0007  80 F9 10                   CMP     CL,16
000A  7C F6                      JL      START
000C  B4 4C                      MOV     AH, 4CH
000E  CD 21                      INT     21H
                                 ;
0010                   DISPLAY:
0010  8A D1                      MOV     DL, CL
0012  80 FA 09                   CMP     DL, 9
0015  7F 06                      JG      DIS1
0017  80 C2 30                   ADD     DL, 48
001A  EB 04 90                   JMP     DIS2
001D  80 C2 37         DIS1:     ADD     DL, 55
0020  B4 02            DIS2:     MOV     AH, 2
0022  CD 21                      INT     21H
0024  C3                         RET
0025                   MAIN      ENDS
                                 END
```

## 서브루틴의 호출과 리턴

위에서 스택은 서브루틴을 호출할 경우에도 쓰인다고 했는데 CPU가 스택을 어떻게 이용하고 있는지를 좀더 자세히 알아보기로 하자.

서브루틴을 호출할 경우 CPU는 돌아올 번지(돌아올 번지는 서브루틴을 호출한 곳의 바로 다음 번지이다)를 스택에 넣고 지정한 서브루틴으로 분기한다. 이 서브루틴에서 또다른 서브루틴으로 분기할 경우 CPU는 다시 돌아올 번지를 스택에 넣고 서브루틴으로 분기를 한다.

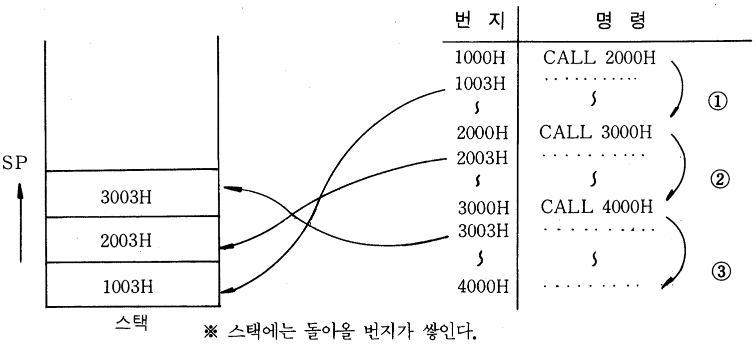
서브루틴에서 다른 서브루틴으로 계속해서 분기를 하면 분기를 할 때마다 되돌아올 번지가 스택에 차곡차곡 쌓이게 된다(그림 1 참조). 그러다가 RET 명령을 만나면 스택의 최상단에 있는 번지를 꺼내서 그 번지로 돌아가게 된다. 이 번지는 맨 나중에 서브루틴을 CALL한 번지이므로 그 곳으로 돌아가게 된다. 그곳에서 RET를 만나면 스택에서 다시 돌아올 번지를 꺼내서 그곳으로 돌아가고, 또 RET를 만나 돌아가는 과정을 되풀이하므로 항상 정확하게 서브루틴을 호출한 시점으로 돌아갈 수 있게 되는 것이다(그림 2 참조).

그러나 경우에 따라서는 왔던 길로 되돌아오지 않고 지름길로 오는 경우가 있듯이 컴퓨터에서도 그런 명령어 체계가 필요하게 된다. 이를테면 서브루틴에서 자신을 호출한 서브루틴으로 돌아가지 않고 곧바로 메인루틴으로 돌아가는 경우 등을 말한다. 'RETURN 100'과 같은 명령이 바로 그것인데 이것은 서브루틴을 호출한 번지로 돌아가지 말고 곧바로 100행으로 가라는 것인데 RETURN 명령과 GOTO 명령의 혼합형이라고 할 수 있다.

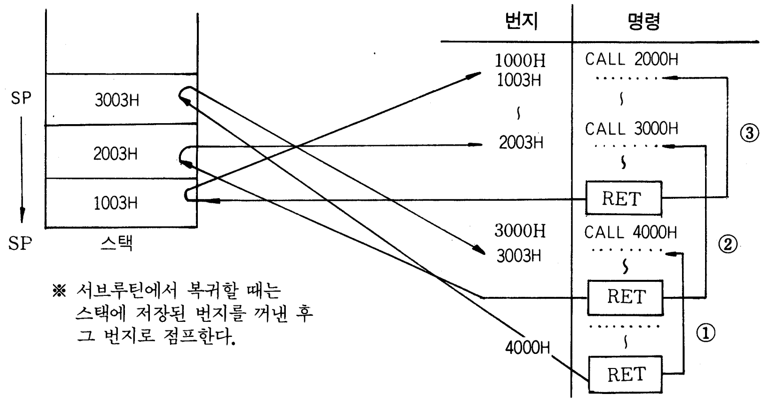
8088도 그와 비슷한 명령 체계를 갖추고 있다. 8088에서는 'RET n'(n은 ± 부호가 붙은 2바이트 수치)과 같이 기술한다. 'RET-2'는 스택에서 2바이트

를 버리고 돌아가는 경우이므로 자신을 호출한 루틴으로 돌아가지 않고 '자신을 호출한 루틴을 호출한 루틴'(표현이 좀 이상하다)으로 돌아가게 된다. 이와같은 것은 어디까지나 고급스런 테크닉에 해당되는 것이지만 스택에 관한 것이니 알아두기 바란다.

〔그림 1〕 서브루틴의 호출과 스택의 동작




〔그림 2〕 RETURN시의 스택의 동작




## 캐리 플래그의 SET과 RESET

캐리 플래그(Carry Flag)는 플래그 레지스터의 각 플래그 중 가장 빈번히 이용하는 플래그이다. 캐리 플래그는 연산의 결과 자리올림(Carry)이나 빌림(Borrow)이 발생한 경우에 1이 된다고 했다.

어셈블리에서는 이런 용도 외에도 캐리 플래그를 어떤 상태의 표시에 이용하기도 한다. 예를 들어 처리과정에서 논리적인 오류가 발생했을 경우 캐리 플래그를 1로 해둔다든지, 디스크로부터 자료를 입출력하는 과정에서 에러가 발생하면 캐리 플래그를 1로 해두는 경우 등이다.

8088에서는 프로그래머가 캐리 플래그를 1 또는 0으로 설정하는 명령이 있다. 이것은 캐리 플래그의 비트를 1로 하느냐 0으로 하느냐의 조작뿐이므로 오퍼랜드를 갖지 않는다.

[형식]
CLC 혹은 STC

CLC(Clear Carry Flag)는 말 그대로 캐리 플래그를 0으로 하는 명령이며, STC(Set Carry Flag)는 캐리 플래그를 1로 하는 명령이다. 캐리 플래그를 상대로 하는 명령은 서브루틴에서 어떤 처리를 한 상태를 메인 프로그램으로 전달하는 경우에 이용된다. 즉, 서브프로그램에서 처리를 제대로 했을 경우에는 캐리 플래그를 0으로 하고 처리에 실패했을 경우에는 캐리 플래그를 1로 하여 메인으로 돌아가게 된다. 메인 프로그램에서는 서부루틴을 호출한 후 캐리 플래그의 상태를 봐서 성공이냐 실패냐의 여부를 판단하게 된다.

예를 들어 키보드로부터 16진수를 입력받는 경우를 생각해보자. 16진수는 0~F사이의 수치로 구성되므로 0~F이외의 문자가 입력되면 입력 에러로 처리해야 한다. 입력된 문자를 판단하는 부분을 서브루틴으로 작성했을 경우에 서브루틴에서는 자료에 0~F 이외의 문자가 있을 경우에는 캐리 플래그를 1로 해서 메인루틴으로 리턴하게 된다.

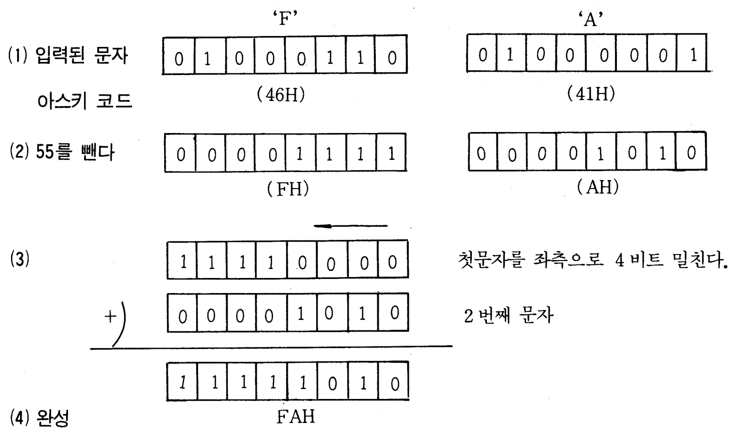
그러면 예제를 가지고 스택에 관한 명령과 캐리 플래그에 의한 상태전달에 대해 익혀보기로 하자.

〈예제 2〉
키보드에서 0~FF의 16진수를 입력받아 그 수치에 해당하는 문자를 화면에 출력하라. 단, 입력하는 값이 null이면 실행을 마치고 그 외에는 반복해서 처리한다.

〈처리 방법〉
입력을 받을 문자는 2문자이지만 이것을 1바이트 수치로 변환하여야 한다. 예로써 F와 A라는 문자를 입력하면 이것을 FAH로 변환해서 FAH에 해당하는 문자를 화면에 출력하여야 한다.

문자를 수치로 변환하는 과정은 리스트 1에서 사용한 알고리즘을 반대로 적용하면 된다. 즉, 입력받은 문자가 0~9 사이의 문자라면 이 문자의 아스키코드에서 48을 빼고, A~F 사이의 문자라면 55를 뺀다. 입력받은 값이 0~9, A~F사이의 값이 아니면 잘못

〔그림 3〕 입력된 문자를 16진 수치로 바꾸는 과정




입력된 문자이므로 에러로 처리해야 한다. (여기서는 편의상 a~f는 처리하지 않기로 한다) 위의 과정을 처리하기 위해서는 모두 4차례의 비교를 해야 한다.

**① 0보다 작으면 에러로 처리(비교 끝)**
**② 9이하이면 0~9의 수치로 처리(비교 끝)**
**③ A보다 작으면 에러로 처리(비교 끝)**
**④ F보다 크면 에러로 처리(비교 끝)**

위의 경우가 아니라면 이 문자는 A~F가 된다.

2개의 문자를 각각 0~F 사이의 수치로 바꾼 다음 앞의 수치를 4비트 좌측으로 밀쳐서 다음의 수치와 더한 값이 원하는 값이 된다. **그림 3**에 이 과정을 보였으니 참고하기 바란다. 비트를 이동하는 명령어는 아직 배우지 않았으므로 4비트를 좌측으로 밀치는 것은 그 수치에 16을 곱해서 처리하기로 한다.

또 한가지, 입력되는 문자가 ↵ 일 경우에는 입력이 종료된 것으로 처리해야 한다. 즉, 맨 처음의 문자가 ↵ 이면 입력된 문자가 없으므로 프로그램을 종료한다.

2번째의 문자가 ↵ 이면 첫문자는 '0', 2번째 문자는 첫번째 문자로 대치한다. 즉, 'F'와 ↵ 를 입력하면 '0F'를 입력한 것으로 처리한다. 이것은 2번째의 문자가 ↵ 이면 첫번째의 문자를 2번째의 문자로 두고 첫번째의 문자는 입력이 없는 것으로 하면 된다.

이와 같은 사항을 고려하여 프로그램을 작성하면 **리스트 2**와 같다. 처리절차가 복잡한 만큼 프로그램이 다소 길고 복잡하다.

지금까지는 프로그램상에서 쓰이는 데이터는 프로그램의 끝에 기술하여 왔는데 리스트 2에서는 반대로 데이터를 프로그램의 선두에 기술하고 있다. 그러므로 프로그램의 실행시 데이터를 건너뛰도록 하기 위해서 'JMP MAIN'을 프로그램 맨 처음에 기술하고 있다.

리스트 2를 순서도로 나타내면 **그림 4**와 같다.

고급언어로는 불과 10~20줄이면 작성할 수 있는 것인데도 어셈블리에서는 이렇듯 길고 복잡하다.

리스트 2를 실행시켜 보면 '0C'를 입력한 것과 ↵ 를 입력한 것의 화면 상태가 다소 다르다. 물론 원하는 캐릭터는 출력되지만 ↵ 를 입력하면 줄바꿈이 일어난다. 원래 ↵ 키의 아스키 코드는 13번(0DH)이고 이것은 CR(Carrage Return)으로 일종의 제어코드인데 CR은 커서를 그 줄의 선두로 옮기는 역할을 한다.

↵ 에 의한 커서의 이동은 리스트 2에서 하는 것이 아니고 키 입력시 DOS에서 하는데 커서가 그 줄의 선두로 이동한 다음 아스키문자를 출력하기 때문에 2번째 문자로 ↵를 입력하면 'Input Data:'라는 메시지와 출력 결과가 겹쳐버린다. 이것을 방지하기 위해 여기서는 2번째의 문자가 ↵ 일 경우에는 1줄을 바꾸도록 하였다.

서브루틴 'CHECK'에서는 입력받은 문자가 0~F 사이의 문자라면 적절히 변환한 다음 리턴을 하고 0~F사이의 문자가 아니면 캐리 플래그를 세트하여 리턴한다. 그러므로 메인 프로그램에서는 문자를 입력받은 후 CHECK루틴을 체크하여 캐리가 나타나면 에러처리를 하게 된다.

# 서브루틴의 구조화

리스트 2를 보면 메인루틴과 서브루틴이 섞여 있어서 주석문을 달아두지 않으면 메인루틴과 서브루틴의 구별이 매우 모호해진다. 어셈블리에서는 서브루틴을 좀 더 구조적이고 알아보기 쉽게하기 위하여 프로시듀어(Procedure) 라는 개념을 도입하였다. 프로시듀어는 다음 형식에 의거 기술한다.

```
〈프로시듀어 이름〉 PROC
        ┌──────────┐
        │ 처리내용 │
        └──────────┘
        RET
〈프로시듀어 이름〉 ENDP
```

이것은 우리가 세그먼트를 작성하는 형식과 흡사하다. 세그먼트의 끝이 ENDS(End Segment)이듯이 프로시듀어는 ENDP(End Procedure)로 끝난다. 메인 프로그램에서의 호출은 서브루틴과 마찬가지로 'CALL 〈프로시듀어 이름〉'으로 한다.

프로시듀어란 고급 언어에서는 주프로그램으로부터 인수를 전달받아 일련의 처리를 한 후 처리 결과를 주프로그램으로 되돌려주는 하나의 단위 프

〔리스트 2〕 0～FF의 16진수를 입력받아 해당문자로 출력하는 프로그램

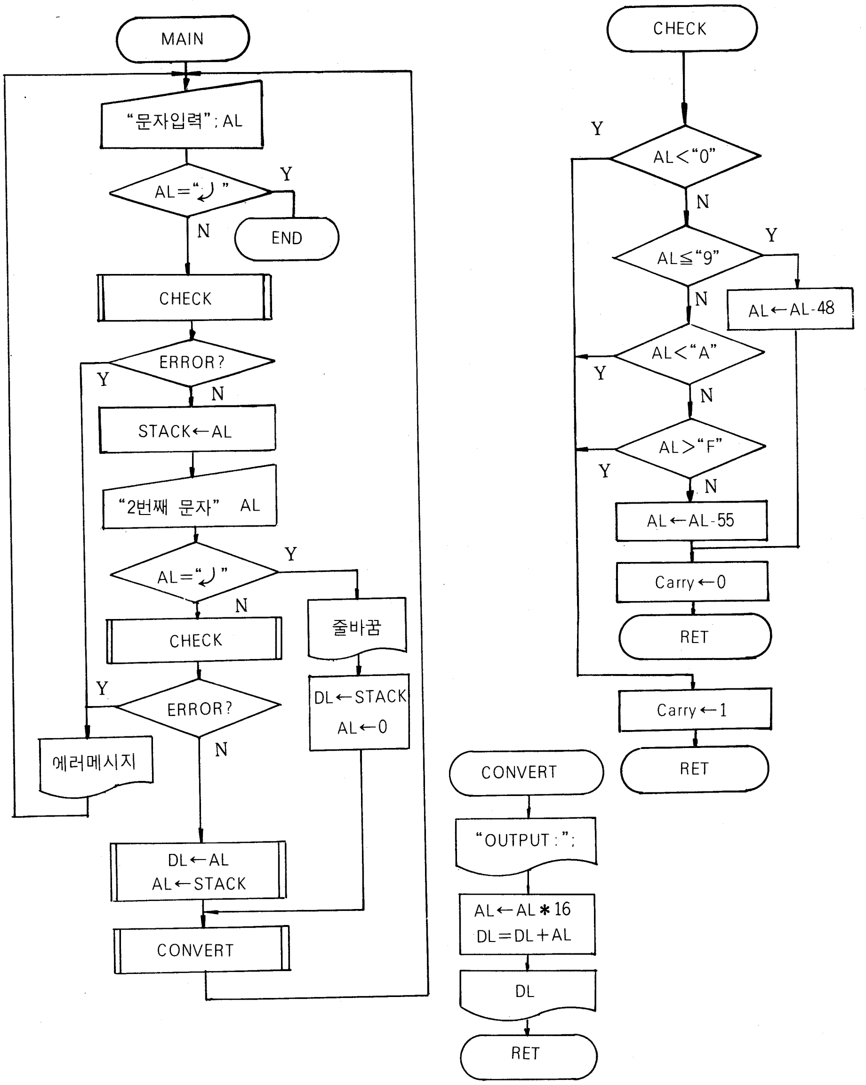
```
0000                          CODE     SEGMENT
                                       ASSUME  CS:CODE, DS:CODE
0000  8C C8                            MOV    AX, CS
0002  8E D8                            MOV    DS, AX
0004  EB 38 90                         JMP    MAIN
0007  49 6E 70 75 74 20 44    DATA     DB     'Input Data : $'
      61 74 61 20 3A 20 24

0015  20 20 3D 3D 3E 20 24    D1       DB     '  ==> $'

001C  50 72 6F 67 72 61 6D    D2       DB     'Program terminated$'
      20 74 65 72 6D 69 6E
      61 74 65 64 24
002F  20 20 44 61 74 61 20    D3       DB     '  Data Error !$'
      45 72 72 6F 72 20 21
      24
003E                          MAIN:    ; --- Print Message ---
003E  B4 09                            MOV    AH,9
0040  BA 0007 R                        MOV    DX, OFFSET DATA
0043  CD 21                            INT    21H
                                       ; --- 1st Charater ---
0045  B4 01                            MOV    AH, 1
0047  CD 21                            INT    21H
0049  3C 0D                            CMP    AL, 13 ; 13 = <Enter> Key
004B  74 24                            JZ     END_PROG
004D  E8 00AD R                        CALL   CHECK
0050  72 2D                            JC     ERROR
0052                          MAIN1:   ; --- 2st Charater ---
0052  50                               PUSH   AX ; Store 1st Charater
0053  B4 01                            MOV    AH, 1
0055  CD 21                            INT    21H
0057  3C 0D                            CMP    AL, 13
0059  75 09                            JNZ    MAIN2
005B  E8 00A2 R                        CALL   LINE_FEED
005E  B0 00                            MOV    AL, 0
0060  5A                               POP    DX
0061  EB 09 90                         JMP    MAIN3
0064  E8 00AD R               MAIN2:   CALL   CHECK
0067  72 16                            JC     ERROR
0069  8A D0                            MOV    DL, AL
006B  58                               POP    AX
006C  E8 008B R               MAIN3:   CALL   CONVERT
006F  EB CD                            JMP    MAIN
                                       ;
0071                          END_PROG:
0071  E8 00A2 R                        CALL   LINE_FEED
0074  BA 001C R                        MOV    DX, OFFSET D2
0077  B4 09                            MOV    AH, 9
0079  CD 21                            INT    21H
007B  B4 4C                            MOV    AH, 4CH
007D  CD 21                            INT    21H
```

```
007F                          ERROR:
                                       ; --- Print Error Messege ---
007F  BA 002F R                        MOV    DX, OFFSET D3
0082  B4 09                            MOV    AH, 9
0084  CD 21                            INT    21H
0086  E8 00A2 R                        CALL   LINE_FEED
0089  EB B3                            JMP    MAIN
                                       ;
                                       ; ---- SUB ROUTINE ----
                                       ;
008B                          CONVERT:
008B  B1 10                   CON1:    MOV    CL, 16
008D  F6 E1                            MUL    CL
008F  02 D0                            ADD    DL, AL
0091  52                               PUSH   DX
0092  BA 0015 R                        MOV    DX, OFFSET D1
0095  B4 09                            MOV    AH, 9
0097  CD 21                            INT    21H
0099  5A                               POP    DX
009A  B4 02                            MOV    AH, 2
009C  CD 21                            INT 21H
009E  E8 00A2 R                        CALL   LINE_FEED
00A1  C3                               RET
                                       ;  ---- Line Feed ----
00A2                          LINE_FEED:
00A2  B4 02                            MOV    AH, 2
00A4  B2 0D                            MOV    DL, 0DH
00A6  CD 21                            INT    21H
00A8  B2 0A                            MOV    DL, 0AH
00AA  CD 21                            INT    21H
00AC  C3                               RET
                                       ;---- Check Input Charater ---
00AD                          CHECK:
00AD  3C 30                            CMP    AL, '0'
00AF  72 15                            JB     ERR
00B1  3C 39                            CMP    AL, '9'
00B3  76 0D                            JBE    CHK1
00B5  3C 41                            CMP    AL, 'A'
00B7  72 0D                            JB     ERR
00B9  3C 46                            CMP    AL, 'F'
00BB  77 09                            JA     ERR
00BD  2C 37                            SUB    AL, 55
00BF  EB 03 90                         JMP    RETURN
00C2  2C 30                   CHK1:    SUB    AL, 48
00C4  F8                      RETURN:  CLC
00C5  C3                               RET
00C6  F9                      ERR:     STC
00C7  C3                               RET
00C8                          CODE     ENDS
                                       END
```

〔그림 4〕 예제 2의 순서도




로그램을 말한다. 그러므로 프로시듀어는 메인 프로그램의 호출이 없으면 결코 스스로는 수행하지 않는다.

그러나 이것은 어디까지나 고급 언어에서의 이야기이고 어셈블리에서는 단순히 서브루틴을 알아보기 쉽게 가시적으로 표현하는 도구에 지나지 않는다. 그러므로 어셈블리에서는 주프로그램이 끝난 다음에도 프로시듀어를 스스로 수행하지 않도록 하는 장치가 필요하다. 프로시듀어는 메인 프로그램의 끝이나 메인 프로그램의 앞에 기술해야 하며, 메인 프로그램의 중간에 기술하면 대부분 프로그램이 폭주하게 된다.

그렇다고 프로시듀어가 단순히 서브루틴을 보기 좋게 하는 겉치레만은 결코 아니다. 뒤에 설명할 모듈별 프로그램을 작성하기 위해서는 반드시 프로시듀어가 필요하고, 또 모듈별로 프로그램을 작성할 때 비로소 프로시듀어의 진가를 발휘하게 된다.

## 프로시듀어의 배치

프로시듀어는 메인 프로그램의 선두에 배치할 수도 있고, 프로그램의 후미에 배치할 수도 있다. 프로시듀어를 어디에 두느냐는 것은 프로그래머의 취향에 따르는 것이지만 프로시듀어를 프로그램의 선두에 둘 경우에는 프로그램의 실행시 프로시듀어를 스스로 실행해버리지 않도록 하는 장치가 필요하다.

이 방법은 리스트 2에서 보인 것과 같이 JMP명령을 사용하여 곧바로 메인 프로그램으로 건너뛰도록 할 수도 있고, 어셈블러에게 프로그램의 시작위치를 알려주는 방법을 이용하기도 한다. 예를 들어 **그림 5**와 같이 프로시듀어가 배치되어 있다고 하자.

그림 5의 프로그램은 MAIN:이라는 레이블이 붙은 번지부터 프로그램을 실행해야 한다. 그러므로 프로그램의 첫머리에 JMP MAIN이라고 기술하면 프로시듀어는 스스로 실행하지 않게 된다.

또다른 방법으로는 맨 나중의 END문을 END MAIN으로 기술하는 것이다. END 다음에 레이블을 기술하면 어셈블러가 어셈블을 한 후에 프로그램의 시작위치를 레이블이 붙은 번지로 설정해 준다. 특히 후자의 방법은 프로시듀어나 데이터를 메인 프로그램의 앞에 배치할 때 정석과 같이 쓰이는 것이므로 잘 익혀두기 바란다.

〈예제 3〉

키보드에서 임의의 문자열을 입력받아 소문자를 대문자로 바꾸어서 출력한다. 입력한 문자열이 null이면 실행을 마치고 그 외에는 반복 처리한다.

〈처리 방법〉

문자열의 입력은 DOS기능 호출의 10번을 이용하기로 한다. 이 기능은 지난 호에서 이미 사용해본 적이 있다. DX에 문자열을 저장할 번지를 설정한 다음 첫머리에 입력받을 최대 문자수를 저장하고 이 기능을 호출하면 다음과 같은 형태로 입력받은 문자열이 저

장되어 돌아온다.

| n | m | 입력한 문자열 | 0DH |
|---|---|---|---|

n　: 입력받을 문자 수
m　: 실제 입력받은 문자 수
0DH : 문자열의 끝을 알리는 딜리미터(delimiter)

입력받은 문자열에서 1문자씩 꺼내 이 문자가 a~z 사이의 문자이면 대문자로 변환하게 된다. 소문자를 대문자로 변환하는 방법은 'a'의 아스키코드가 97, 'A'의 아스키코드가 65이므로 소문자의 아스키코드에서 32(97−65=32) 를 빼면 된다(리스트 3 참조).

리스트 3은 프로시듀어를 메인프로그램의 앞에 배치하였다. 그러므로 프로그램의 맨 끝의 END에 MAIN을 기술하여 프로그램이 MAIN부터 시작함을 알려야 한다. 이와 같은 기법은 프로그램상에서 자주 쓰이는 것이므로 잘 익혀두어야 할 것이다.

## 모듈별 프로그래밍

대부분의 어셈블러는 모듈별 프로그래밍을 할 수 있는 환경을 지원한다. 여기서 모듈(module)이란 '하나의 단위'를 나타내는 것으로 최소한의 처리를 하는 일련의 프로그램을 말한다. 리스트 3에 나타난 프로시듀어들이 바로 하나의 모듈이다.

실제로 제법 쓸만한 프로그램을 작성하려면 소스프로그램은 당연히 길어지게 마련인데 그 프로그램을 기능 별로 프로시듀어를 만들어 가다보면 예전에 사용한 적이 있는 프로시듀어도 있기 때문이다. 만약에 그 프로시듀어가 독립된 화일로 디스크에 수록되어 있다면 현재의 소스프로그램에 그 화일을 삽입한 후에 어셈블을 할 수도 있고, 각각의 모듈을 어셈블하여 링크할 때 결합할 수도 있다.

전자의 방법은 프로그램이 비교적 간단하고 사이즈가 작은 경우에 이용되며, 후자는 프로그램이 크고 복잡할 경우에 이용된다. 먼저 전자의 방법에 대해서 알아보자.

어셈블리에서는 INCLUDE 〈화일명〉에 의해 디스크의 내용을 소스프로그램에 삽입한다. INCLUDE는 기계어가 아니고 의사명령 중의 하나이다. 예로써 리스트 3의 INPUT 프로시듀어가 독립된 화일로 디스크에 INPUT.INC로 저장되어 있다면 리스트 3에서는 INPUT 프로시듀어를 기술할 자

〔그림 5〕 프로시듀어 배치 예

```
〈세그먼트〉   SEGMENT
        ┌──────────────┐
        │  프로시듀어 1  │
        └──────────────┘
        ┌──────────────┐
        │  프로시듀어 2  │
        └──────────────┘
        ┌──────────────┐
        │  프로시듀어 3  │
        └──────────────┘
MAIN :
        ┌──────────────┐
        │  메인 프로그램  │
        └──────────────┘
〈세그먼트〉   ENDS
             END
```

〔리스트 3〕 입력받은 문자열을 대문자로 바꾸어 준다.

```
0000                          CODE    SEGMENT
                                      ASSUME   CS:CODE, DS:CODE
0000  49 6E 70 75 74 20 20    MESS1   DB  'Input  String : $'
      53 74 72 69 6E 67 20
      3A 20 24
0011  4F 75 74 70 75 74 20    MESS2   DB  'Output String : $'
      53 74 72 69 6E 67 20
      3A 20 24
0022  50 72 6F 67 72 61 6D    MESS3   DB  'Program Terminated.$'
      20 54 65 72 6D 69 6E
      61 74 65 64 2E 24
0036  00FF[                   STRING  DB  255  DUP(0)
          00
                    ]

0135                          INPUT     PROC
                                      ;    --- Input  String ---
0135  BB 0036 R                       MOV   BX, OFFSET STRING
0138  C6 07 FF                        MOV   BYTE PTR [BX], 255
013B  BA 0000 R                       MOV   DX, OFFSET MESS1
013E  B4 09                           MOV   AH, 9
0140  CD 21                           INT   21H
                                      ;
0142  BA 0036 R                       MOV   DX, OFFSET STRING
0145  B4 0A                           MOV   AH, 10
0147  CD 21                           INT   21H
0149  E8 018C R                       CALL  CRLF
014C  C3                              RET
                              INPUT   ENDP
                                      ;
014D                          CONVERT  PROC
                                      ;    --- Convert Lower-case to Upper-c
                              ase
014D  F8                              CLC
014E  BB 0038 R                       MOV   BX, OFFSET STRING+2
0151  8A 0E 0037 R                    MOV   CL, STRING+1
0155  80 F9 00                        CMP   CL, 0
```

```
0158  75 02                 JNZ   REPEAT
015A  F9                    STC
015B  C3                    RET
015C  8A 07        REPEAT:  MOV   AL, [BX]
015E  3C 61                 CMP   AL, 'a'
0160  72 08                 JB    CON1
0162  3C 7A                 CMP   AL, 'z'
0164  77 04                 JA    CON1
0166  2C 20                 SUB   AL, 32
0168  88 07                 MOV   [BX],AL
016A  43            CON1:   INC   BX
016B  FE C9                 DEC   CL
016D  75 ED                 JNZ   REPEAT
016F  C6 07 0D              MOV   BYTE PTR [BX], 0DH
0172  C6 47 01 0A           MOV   BYTE PTR [BX+1], 0AH
0176  C6 47 02 24           MOV   BYTE PTR [BX+2], '$'
017A  F8                    CLC
017B  C3                    RET
                   CONVERT  ENDP
                            ;
017C               OUTPUT   PROC
                            ;  --- Output String ---
017C  BA 0011 R             MOV   DX, OFFSET MESS2
017F  B4 09                 MOV   AH, 9
0181  CD 21                 INT   21H
0183  BA 0038 R             MOV   DX, OFFSET STRING+2
0186  CD 21                 INT   21H
0188  E8 018C R             CALL  CRLF
018B  C3                    RET
                   OUTPUT   ENDP
                            ;
018C               CRLF     PROC
                            ;  --- Line feed ---
018C  B2 0D                 MOV   DL, 0DH
018E  B4 02                 MOV   AH, 2
0190  CD 21                 INT   21H
0192  B2 0A                 MOV   DL, 0AH
0194  CD 21                 INT   21H
0196  C3                    RET
                   CRLF     ENDP
                            ;
0197               MAIN:    ; --- Main Program ---
0197  8C C8                 MOV   AX, CS
0199  8E D8                 MOV   DS, AX
019B  E8 0135 R             CALL  INPUT
019E  E8 014D R             CALL  CONVERT
01A1  72 05                 JC    END_PROG
01A3  E8 017C R             CALL  OUTPUT
01A6  EB EF                 JMP   MAIN
01A8               END_PROG:
01A8  BA 0022 R             MOV   DX, OFFSET MESS3
01AB  B4 09                 MOV   AH, 9
01AD  CD 21                 INT   21H
01AF  B4 4C                 MOV   AH, 4CH
01B1  CD 21                 INT   21H
01B3               CODE     ENDS
                            END   MAIN
```

리에 다음과 같이 기술하면 된다.

```
INCLUDE A:INPUT.INC
```

또, 위의 리스트 3에서 쓰인 INPUT 프로시듀어와 CONVERT, CRLF 프로시듀어는 다른 프로그램에서도 쓰일 수 있을 것이다.

DOS에서는 소문자로 입력해도 자동으로 대문자로 치환해 주지만 어셈블리 프로그램을 작성할 경우에는 우리 스스로 처리해야 하므로 CONVERT 프로시듀어가 다른 프로그램에서 쓰일 수 있는 가능성이 크다.

그러므로 이 프로시듀어만을 따로 떼어내서 INPUT.INC와 CONVERT.INC, CRLF.INC로 디스크에 수록해두기 바란다. 예제 프로그램에서 문자열을 입력받고, 또 소문자를 대문자로 변환할 필요성이 있을 경우에는 곧바로 이 프로시듀어들을 소스상에 삽입하여 처리할 수 있게 하기 위해서이다.

각각의 모듈을 따로 어셈블하여 링크하는 경우에는 다소 절차가 복잡하다. 주프로그램을 A, 부프로그램을 B라고 두고 설명하기로 하자. 먼저 B를 설계한다.

```
<프로그램 B>
<세그먼트> SEGMENT PUBLIC
ASSUME CS:<세그먼트>, DS:<세그먼트>
PUBLIC <프로시듀어 1>, <프로시듀어 2>,
…
<프로시듀어 1> PROC NEAR
<프로시듀어 1> ENDP
<프로시듀어 2> PROC NEAR
<프로시듀어 2> ENDP
<프로시듀어 3> PROC NEAR
<프로시듀어 3> ENDP
<세그먼트> ENDS
           END
```

부프로그램에서는 프로그램의 세그먼트 다음에 PUBLIC이라고 기술해야 하는데 PUBLIC이라는 것은 이 모듈이 다른 모듈에서 참조될 수 있음을 나타내는 의사명령이다.

ASSUME문 다음의 행에는 PUBLIC을 기술한 후 다른 모듈에서 참조할 프로시듀어 이름을 쓰는데, 모든 프로시듀어를 다 기술할 필요는 없고 외부

에서 CALL로 호출될 프로시듀어만 기술한다. 그러므로 프로그램 B의 내부에서 호출되는 프로시듀어는 기술하지 않아도 된다.

위에서 프로시듀어 이름 옆에 NEAR라고 기술하였는데 NEAR는 그 프로시듀어가 메인프로그램과 동일한 세그먼트(코드 세그먼트)에 배치된다는 것을 나타낸다. 실제로 리스트 3에서도 각 프로시듀어에 NEAR를 붙여야하는 것이지만 위에서는 생략을 하였다. 프로그램 B가 프로그램 A와 세그먼트(코드 세그먼트)가 다를 경우에는 NEAR 대신에 FAR를 붙이게 되는데 이것에 대해서는 장을 바꿔서 설명하기로 한다. 다음에는 프로그램 A를 설계한다.

```
〈프로그램 A〉
〈세그먼트〉 SEGMENT PUBLIC
  ASSUME CS:〈세그먼트〉, DS:〈세그먼트〉
  EXTRN 〈프로시듀어 1〉:NEAR,
  〈프로시듀어 2〉:NEAR…
  CALL 〈프로시듀어〉
  CALL 〈프로시듀어〉
〈세그먼트〉 ENDS
           END
```

주프로그램에서도 프로그램의 첫머리에 PUBLIC를 기술한다. 그 다음에는 세그먼트의 정의를 하고나서 EXTRN(External)을 기술한 후 〈프로시듀어 이름〉을 나열하는데 EXTRN은 그 프로시듀어가 외부에서 정의되어 있다는 것을 명시하는 의사명령이다.

또, 〈프로시듀어 이름〉 다음에는 NEAR 나 FAR를 붙여서 그 프로시듀어가 동일한 세그먼트인지, 세그먼트가 다른지를 명시해야 한다. FAR에 대해서는 장을 달리하여 설명하기로 했으므로 여기서는 반드시 NEAR를 붙이는 것으로 한다.

프로그램을 모듈별로 작성할 경우에는 각각의 모듈을 독립된 프로그램의 형식으로 하되 모듈간의 〈세그먼트〉 이름이 같도록 해야 한다. 또한 데이터를 다루지않는 모듈에서는 ASSUME으로 DS를 선언하지 않도록 한다. 또 한가지 메인 프로그램에서는 다음의 문장을 프로그램의 첫머리에 두어야 한다.

```
MOV  AX,CS
MOV  DS,AX
```

이것은 데이터 세그먼트를 코드 세그먼트와 일치시키는 절차인데 만일 데이터 세그먼트를 따로 설정할 경우에는 거기에 따른 방법으로 DS를 설정해야 한다. 이것은 메인 프로그램에 데이터가 있든지 없든지간에 DS의 설정은 반드시 메인측에서 한다고 알고 있으면 된다.

프로그램의 완성이 되었으면 각각 어셈블한다. (링크는 한꺼번에 할 것이므로 따로 링크하지 않도록 한다.) 다음에는 링크를 한다.

```
A>LINK ↲
Object Modules [.OBJ]:〈프로그램 A〉+
〈프로그램 B〉
Run File [〈프로그램 A〉.EXE]: ↲
List File [Null.MAP]: ↲
Libraries [.LIB]: ↲
```

모듈들을 결합하여 링크할 경우에는 모듈들을 '+'로 연결해서 기술한다. 이때 메인 프로그램을 반드시 선두에 기술하지 않으면 안된다.

프로그램을 독립적으로 어셈블한 후에 링크시키는 방법은 프로그램을 디버깅하는데 많은 어려움이 뒤따른다. 또한 프로그램을 고칠 경우에는 각각의 루틴을 수정해야 하기때문에 프로그램을 합쳐서 통째로 만드는 것보다 더 많은 수고가 필요하다. 그러나 일단 독립모듈을 만들어 두기만 하면 다른 프로그램과도 곧바로 링크하여 사용할 수 있으므로 독자적인 라이브러리(서브루틴의 모음)를 구축할 수 있게 된다. 그러면 예제를 가지고 분할 어셈블하는 경우를 실습해보기로 한다.

〈예제 4〉

특정한 키를 입력하여 화면에 윈도우(Window)를 표시, 제거하는 프로그램을 다음의 조건에 의거 작성하라.

〈A〉: 윈도우의 표시(윈도우는 역상 문자로 표시한다.)
〈B〉: 윈도우의 메시지를 위로 1줄 스크롤
〈C〉: 윈도우의 메시지를 아래로 1줄 스크롤
〈ESC〉: 윈도우의 제거(화면 복구)
〈X〉: 프로그램의 종료

〈처리 방법〉

언뜻 보기에는 고도의 테크닉이 필요한 프로그램처럼 보인다. 그러나 원리를 알면 별로 대단한 것은 아니다.

이 강좌의 첫회에서 'INT 21H'는 일종의 인터럽트 명령이라고 한 적이 있다. 또 인터럽트 명령은 BIOS 인터럽트와 DOS 인터럽트가 있다고 하였다. IBM-PC에는 이런 인터럽트가 0H~FFH까지 255가지가 있으며 이들의 기능은 각각 다르고, 그 중에는 사용하지 않는 것들도 있다. 이들 중 21H 인터럽트가 바로 우리가 사용해온 DOS 기능호출이다.

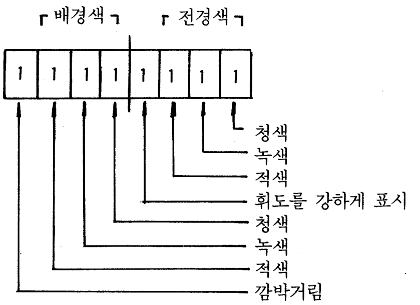
화면에 윈도우를 출력하기 위해서는 10H 인터럽트를 이용한다. INT 10H는 주로 화면제어에 이용된다. INT 10H는 다시 0~15번까지 있는데 여기서 이용하는 것은 6번과 7번이다. 6번의 기능은 다음과 같다.

INT 10H 6번 : 화면을 위로 스크롤한다.
AL : 스크롤 할 line 수 (0이면 전체를 지운다.)
CH : Window의 좌측 상단 row 위치
CL : Window의 좌측 상단 column
DH : Window의 우측 하단 row
DL : Window의 우측 하단 column
BH : Window의 배경 컬러 코드
AH : 6(기능번호)

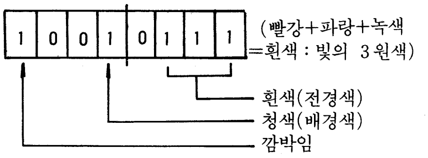
7번의 기능은 화면을 아래로 스크롤

한다는 것만 다를 뿐 기능은 동일하다. 위에서 Window라는 표현은 필자가 임시로 붙인 것으로 화면의 특정한 구역을 가리킨다. 화면전체를 지울 경우에는 윈도우의 모서리를 (0, 0)과 (24, 79)로 지정하고 스크롤할 줄수를 0으로 설정하면 된다. IBM-PC의 화면 구성은 **그림 6**과 같다.

배경 컬러코드는 화면을 지울 때 역상모드로 하느냐 정상모드로 하느냐 등을 나타내는 것으로 1바이트 수치로 표시하는데 각각의 비트는 다음의 색상을 표시하게 된다.




또, 각 비트끼리 조합하여 여러가지 색상을 만들 수 있다. '파란 바탕에 깜박이는 흰 글자'는 다음과 같은 비트 패턴(97H)이 된다.




모노크롬에서는 맨 우측의 비트가 밑줄문자를 표시하는데 쓰이며 각각의 컬러값은 항상 흰색으로 대치된다. 역상문자를 만들기 위해서는 전경색의 각 비트를 0으로 두면 된다. 화면에 윈도우를 표시하기 위해서는 다음과 같은 절차를 거쳐야 한다.

**① 화면상에서 윈도우가 표시될 부분에 출력되어 있는 내용을 메모리로 대피한다.**

**② 윈도우를 화면에 출력한다.**

**③ ………**

**④ 메모리에 대피한 화면의 내용을 복구한다.**

화면을 메모리로 대피하기 위해서는 먼저 컴퓨터가 화면 출력을 어떻게 하는지를 알아야 한다. 컴퓨터의 메모리에는 스크린 버퍼로 할당된 영역이 있는데 허큘리스 보드의 경우 절대번지가 B0000H이며, 그래픽보드의 경우에는 B8000H로 되어 있다. 스크린 버퍼는 화면과 2 : 1로 대응되며 스크린 버퍼에 문자의 아스키코드와 그 속성을 넣어주면 해당 위치에 그 문자가 출력되게 된다. 스크린 버퍼는 80(컬럼)＊25 (줄)＊2(아스키코드, 속성)=4000바이트의 크기인데 **그림 7**과 같이 매핑된다.

화면에 윈도우를 출력하는 방법은 먼저 윈도우에 출력할 문자열을 메모리에 저장하여 두어야 한다. 이 문자열을 화면에 표시하기 위해서는 INT 21H의 문자열 출력루틴(9번)을 이용할 수도 있으나 이 경우에는 커서의 위치를 지정해야할 뿐 아니라 화면의 출력속도가 무척 떨어지게 된다.

화면에 윈도우를 신속하게 열고 닫고 하려면 메모리의 내용을 직접 스크린 버퍼로 전송하는 방법을 이용한다. 윈도우는 **그림 8**과 같이 구성할 것이므로 윈도우의 첫머리번지를 계산하면 B0000H로부터 500바이트가 떨어진 곳(오프셋)이 된다. 글자수로 계산하면 80＊3(줄)+10(컬럼)=250(문자)이지만 각 글자마다 속성이 있어야 하므로 250＊2=500이 되는 것이다. 또 윈도우는 50(column)＊10(row)이지만 테두리가 있어야 하므로 실제로는 52＊12의 크기가 된다.

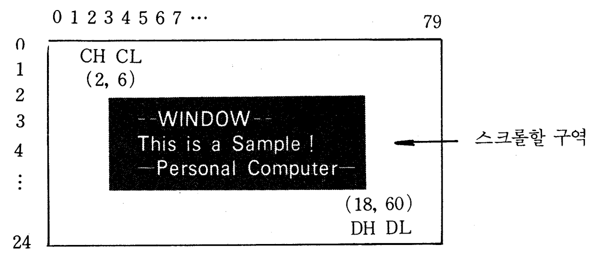
화면에서 키를 입력받는 것은 INT 21H의 7번 기능을 이용하기로 한다. 7번의 기능은 입력받은 문자를 화면에 표시하지 않는다는 것만 다를 뿐 1번과 비슷하다.

이와같은 사항을 염두에 두고 프로그램을 작성하면 **리스트 4, 리스트 5**와 같다. 리스트 4는 주프로그램으로 키입력을 판단하여 각각의 프로시듀어를 호출하는 기능만 하고 해당 프로시듀어는 리스트 5에 배치하였다.

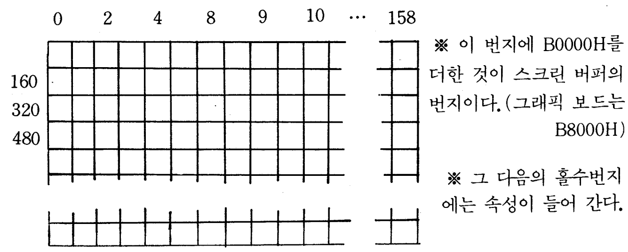
프로그램의 실행방법은 리스트 4와 리스트 5를 각각 입력하여 디스크에 'LIST-4.ASM', 'LIST-5.ASM'으로 세이브한 다음 **화면 1**의 과정을 거쳐서 어셈블, 링크하면 된다. 이때 리스트 5의 작은 따옴표(')로 묶여있는 문자열의 길이가 작은 따옴표를 제외하고 정확하게 50문자가 되도록 하여야 한다. 글자 수가 다르면 윈도우의 테두리가 제대로 출력되지 않으므로 주의하여야 한다.

리스트 5를 보면 다음과 같은 문장

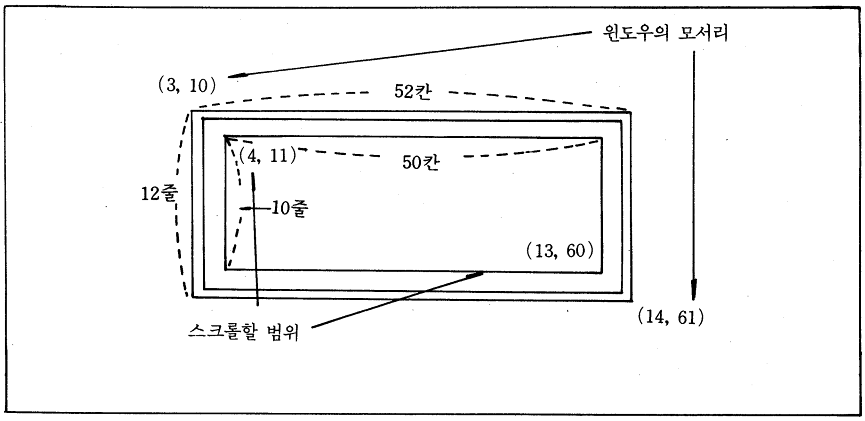
〔그림 6〕 스크롤할 IBM-PC의 화면구성




〔그림 7〕 스크린 버퍼의 매핑



〔그림 8〕 윈도우의 구성




이 있다.

```
MOV BYTE PTR ES:[BX], 70H
```

이것은 ES 레지스터의 값을 기준으로 하고 BX 레지스터의 값을 오프셋으로 한 번지에 70H를 저장하라는 의미이다.

8088에서는 이와 같이 16비트로는 표현할 수 없는 번지는 세그먼트 레지스터와 옵셋을 조합하여 표시하게 된다. 리스트 5를 참고로 프로그램논리를 이해하기 바란다.

〔리스트 4〕 예제 4의 메인 프로그램

```
0000               CODE     SEGMENT   PUBLIC
                            ASSUME  CS:CODE
                   EXTRN  PUT_WINDOW:NEAR, SCR_UP:NEAR, SCR_DOW
                   N:NEAR, CLS_WINDOW:NEAR
0000  8C C8                 MOV    AX, CS      ; Set  DS
0002  8E D8                 MOV    DS, AX
                            ;
0004  B4 07        MAIN:    MOV    AH, 7       ; 1문자 입력
0006  CD 21                 INT    21H
0008  3C 58                 CMP    AL, 'X'
000A  74 24                 JZ     END_PROG
000C  3C 41                 CMP    AL, 'A'
000E  75 05                 JNZ    MA_1
0010  E8 0000 E             CALL   PUT_WINDOW
0013  EB EF                 JMP    MAIN
0015  3C 42          MA_1:  CMP    AL, 'B'
0017  75 05                 JNZ    MA_2
0019  E8 0000 E             CALL   SCR_UP
001C  EB E6                 JMP    MAIN
001E  3C 43          MA_2:  CMP    AL, 'C'
0020  75 05                 JNZ    MA_3
0022  E8 0000 E             CALL   SCR_DOWN
0025  EB DD                 JMP    MAIN
0027  3C 1B          MA_3:  CMP    AL, 27      ; 27 = <ESC>
0029  75 D9                 JNZ    MAIN
002B  E8 0000 E             CALL   CLS_WINDOW
002E  EB D4                 JMP    MAIN
0030               END_PROG:
0030  B4 4C                 MOV    AH, 4CH
0032  CD 21                 INT    21H
0034               CODE     ENDS
                            END
```

〔리스트 5〕 예제 4의 프로시듀어 프로그램

```
CODE       SEGMENT  PUBLIC
           ASSUME   CS:CODE, DS:CODE
  PUBLIC   PUT_WINDOW,SCR_UP,SCR_DOWN,CLS_WINDOW
PUT_WINDOW    PROC    NEAR
        MOV    AH, FLAG                 ; 이미 윈도우가 열려있으면
        CMP    AH, 0                    ;   메인으로 돌아간다.
        JZ     PUT_1
        RET
PUT_1:  MOV    AH, 1                    ; FLAG에 1을 저장하여 윈도우가
        MOV    FLAG, AH                 ;   열려있다는 것을 알린다.
        CALL   SAVE_SCRN                ; 화면을 대피하는 루틴을 호출
        MOV    AX, 0B000H               ; 스크린 버퍼의 선두를 ES 레지스터에
        MOV    ES, AX                   ;   넣어서 세그먼트 취급을 한다.
        MOV    BX, 500                  ; 윈도우의 좌측 모서리가 위치할
        MOV    BP, OFFSET WINDOW        ;   스크린 버퍼의 번지
        MOV    CX, 12                   ; 화면으로 전송할 줄 수
        MOV    DI, 52                   ; 화면으로 전송할 컬럼 수
PUT_2:  PUSH   DI                       ; DI, BX, CX를 스택에 대피
        PUSH   BX                       ;  (CX와 DI는 반복 횟수의 카운트에
        PUSH   CX                       ;  이용된다)
PUT_3:  MOV    AH, [BP]                 ; 메모리의 내용을 1바이트 스크린
        MOV    ES:[BX],AH               ;   버퍼에 전송
        INC    BX                       ; 문자의 속성(70H=역상 문자)을 스크린
        MOV    BYTE PTR ES:[BX],70H;      버퍼에 전송
        INC    BX                       ; 스크린 버퍼의 번지와 메모리의 번지를
        INC    BP                       ;   1바이트 분 이동
        DEC    DI                       ; 52컬럼 분이 처리되지 않았으면 PUT_3
        JNZ    PUT_3                    ;   으로 올라가서 반복처리한다.
        POP    CX                       ; 레지스터의 회복
        POP    BX                       ;
```

```
        POP     DI                  ;
        ADD     BX, 160             ; 1줄 아래로 이동한다.
        DEC     CX                  ; 12줄을 모두 처리하지 않았으면
        JNZ     PUT_2               ;   PUT_2로 올라가 반복한다.
        RET
PUT_WINDOW      ENDP
SAVE_SCRN       PROC  NEAR
        MOV     DI, OFFSET  DATA    ; DI = 화면을 저장할 메모리 번지
        MOV     AX, 0B000H          ; 스크린 버퍼의 번지를 ES에 넣어
        MOV     ES, AX              ;   세그먼트 취급을 한다.
        MOV     CX, 12              ; 메모리에 저장할 줄 수
        MOV     DX, 104             ; 1줄당 바이트 수(52자*2=104)
        MOV     BX, 500             ; 윈도우 좌측 상단의 화면상의 번지
SA1:    PUSH    CX                  ; 레지스터의 대피
        PUSH    DX                  ;
        PUSH    BX                  ;
SA2:    MOV     AL, ES:[BX]         ; 화면의 내용을 1바이트 메모리로 대피
        MOV     [DI], AL            ;   한다
        INC     DI                  ;
        INC     BX                  ;
        DEC     DX                  ; 104바이트(52문자)를 처리하지 못했
        JNZ     SA2                 ;   으면 SA2로 간다.
        POP     BX
        POP     DX
        POP     CX
        ADD     BX, 160             ; 1줄 내려간다.
        DEC     CX                  ; 12줄을 처리하지 못했으면 SA1로 가서
        JNZ     SA1                 ;   반복처리 한다.
        RET
SAVE_SCRN   ENDP
SCR_UP      PROC
        MOV     AH, FLAG            ; 윈도우가 표시되어있지 않으면
        CMP     AH, 0               ; 메인루틴으로 돌아간다.
        JNZ     UP_1
        RET
UP_1:   MOV     CX, 040BH           ; CH:04 CL:11 (스크롤할 좌측 코너)
        MOV     DX, 0D3CH           ; DH:13 DL:60 ( " 우측 "  )
        MOV     BH, 70H             ; 배경색의 컬러 (역상 문자)
        MOV     AL, 1               ; 스크롤할 줄 수
        MOV     AH, 6               ; 위로 스크롤
        INT     10H
        RET
SCR_UP      ENDP

SCR_DOWN    PROC
        MOV     AH, FLAG            ; 윈도우가 표시되어있지 않으면
        CMP     AH, 0               ; 메인루틴으로 돌아간다.
        JNZ     DN_1
        RET
DN_1:   MOV     CX, 040BH           ; CH:04 CL:11 (스크롤할 좌측 코너)
        MOV     DX, 0D3CH           ; DH:13 DL:61 ( "  우측  "  )
        MOV     BH, 70H             ; 배경색의 컬러 (역상 문자)
        MOV     AL, 1               ; 스크롤할 줄 수
        MOV     AH, 7               ; 아래로 스크롤
        INT     10H
        RET
SCR_DOWN    ENDP

CLS_WINDOW   PROC  NEAR
        MOV     AH, FLAG            ; 윈도우가 표시되어있지 않으면
        CMP     AH, 0               ; 메인루틴으로 돌아간다.
        JNZ     CLS
        RET
CLS:    MOV     AH, 0               ; 윈도우 표시용 플래그를 리세트
        MOV     FLAG, AH            ;
        MOV     DI, OFFSET  DATA
        MOV     AX, 0B000H          ;
        MOV     ES, AX              ; 메모리에 저장된 내용을 화면으로
        MOV     CX, 12              ;   되돌리는 부분은 화면의 내용을
        MOV     DX, 104             ;   메모리로 저장하는 부분과 레지
        MOV     BX, 500             ;   스터만 다를 뿐 거의 동일하다.
CL1:    PUSH    CX
        PUSH    DX
        PUSH    BX
CL2:    MOV     AL, [DI]
        MOV     ES:[BX], AL
        INC     DI
        INC     BX
        DEC     DX
        JNZ     CL2
        POP     BX
        POP     DX
        POP     CX
        ADD     BX, 160
        DEC     CX
        JNZ     CL1
        RET
CLS_WINDOW    ENDP
FLAG      DB  0
WINDOW    DB 201, 50 DUP(205), 187
          DB 186, 50 DUP(32), 186
          DB 186, 'This is a sample  Program  to make Window.',186
          DB 186, '                                          ',186
          DB 186, '         W I N D O W !                    ',186
          DB 186, '                                          ',186
          DB 186, '                H E L P !                 ',186
          DB 186, '                                          ',186
          DB 186, '     -- Personal   Computer --            ',186
          DB 186, '                                          ',186
          DB 186, 50 DUP(32),186
          DB 200, 50 DUP(205),188
DATA      DB 1500  DUP (0)
CODE      ENDS
          END
```

〔화면 1〕 리스트 4와 리스트 5의 어셈블과 링크

```
C:\MASM>MASM
Microsoft (R) Macro Assembler  Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985.  All rights reserved.

Source filename [.ASM]: LIST-4
Object filename [LIST-4.OBJ]: LIST-4
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51060 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\MASM>MASM
Microsoft (R) Macro Assembler  Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985.  All rights reserved.

Source filename [.ASM]: LIST-5
Object filename [LIST-5.OBJ]: LIST-5
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  49736 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\MASM>LINK

Microsoft (R) 8086 Object Linker  Version 3.05
Copyright (C) Microsoft Corp 1983, 1984, 1985.  All rights reserved.

Object Modules [.OBJ]: LIST-4+LIST-5
Run File [LIST-4.EXE]: LIST-4
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: no stack segment

C:\MASM>
```

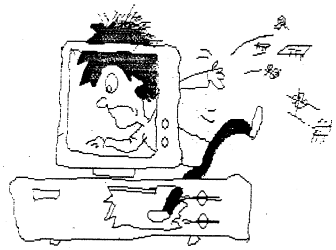
# 결어

이번호에는 조금 이른 감이 있지만 다른 인터럽트를 사용해서 프로그램을 작성해 보았다. INT 10H는 화면 제어용으로 많이 쓰이는 인터럽트로 커서의 위치를 결정한다거나 화면에 점, 선 등의 도형을 그릴 때도 이용된다.

아직은 독자들이 명령어를 배우는 단계이므로 실생활에 쓰일 수 있는 프로그램을 예제로 선택하기 어려운 감이 있으나 가능하다면 고급언어로 처리하기 어려운 부분들을 예제로 선택하려고 한다.

다음 호에서는 비트 연산에 관한 명령과 반복처리를 쉽게하는 명령에 대해서 설명할 예정이다. 본 강좌에 대해서 의문나는 점이나 개선해야 할 점이 있으면 필자의 주소로 연락해 주기 바란다.(필자의 사정상 전화는 사양) PC

㉾ 469-800
경기도 여주군 여주읍 창리 150-3
여주 컴퓨터 학원 박승제



첨단서적

# 컴퓨터 전문서적 통신판매 써비스

| | | |
|---|---|---|
| 표준 MS DOS 핸드북 / 기전연구사 7,000원 | 알기 쉬운 dBASE Ⅲ / 영진 6,000원 | SIDEKICK plus / 성안당 6,000원 |
| MS DOS 핸드북 / 정보시대 3,000원 | dBASE Ⅲ plus 프로그램 모음집 / 영진 9,000원 | GW BASIC 3.2 / 성안당 4,500원 |
| MS DOS 입문 / 성안당 5,000원 | dBASE Ⅲ plus / 크라운 7,000원 | GW BASIC / 크라운 5,500원 |
| MS DOS / 성안당 5,000원 | dBASE Ⅲ plus / 기전연구사 6,000원 | IBM BASIC / 성안당 6,000원 |
| IBM DOS 3.2 / 크라운 5,000원 | Auto CAD 2.6 / 영진 15,000원 | 워드 퍼펙트 / 우진출판사 7,000원 |
| MS DOS 3.3 / 크라운 6,000원 | Auto CAD 실무 / 크라운 5,500원 | PC TOOLS DELUXE / 크라운 3,000원 |
| 알기 쉬운 MS DOS / 영진 5,000원 | Auto CAD 실습과 응용 / 영진 10,000원 | 터보파스칼 4.0 / 크라운 8,000원 |
| 고급과정 MS DOS / 크라운 6,000원 | C언어 기초+α / 교학사 8,000원 | WORDSTAR 4.0 / 우진 6,000원 |
| MS DOS 핸드북 / 크라운 9,000원 | Run C / 교학사 7,000원 | WORDSTAR 실무 / 영진 4,500원 |
| DOS 3.3 / 영진 8,000원 | C언어의 이론 응용 / 성안당 7,000원 | 애플 Ⅱ 입문 / 영진 3,500원 |
| MS DOS 4.0 / 연학사 6,500원 | C 프로그래밍 / 동일 5,000원 | Symphony 메뉴얼 / 크라운 7,000원 |
| MS DOS 메뉴얼 / 크라운 4,000원 | C언어 500제 / 크라운 5,000원 | Symphony 메뉴얼 / 영진 7,000원 |
| MS DOS와 GW BASIC / 연학사 5,500원 | C언어 입문과 실습 / 크라운 5,500원 | RM COBOL / 성안당 5,000원 |
| Lotus 123 기본 / 크라운 5,000원 | C 라이브러리 / 영진 5,000원 | MS COBOL / 성안당 6,000원 |
| 실무 Lotus / 크라운 6,000원 | TURBO C(User's) / 크라운 5,000원 | IBM PC 어셈블리 / 기전연구사 6,000원 |
| Lotus Leal 2 / 크라운 5,000원 | TURBO C(Reference) / 크라운 5,000원 | IBM PC 하드웨어 종합분석 / 성안당 5,500원 |
| 완벽 Lotus / 영진 5,000원 | MS C 교과서 / 성안당 6,000원 | MSX DOS 입문 / 연학사 5,500원 |
| 알기 쉬운 Lotus / 크라운 9,000원 | Clipper Compiler / 영진 5,000원 | MSX DOS 슈퍼 핸드북 / 우진 4,500원 |
| Lotus 123 / 기전연구사 6,000원 | 실무자를 위한 Crosstalk / 영진 5,500원 | MSX BASIC 컴퓨터교실 / 기전연구사 4,000원 |
| 실무 Lotus 123 / 영진 10,000원 | 노턴 유틸리티 / 영진 3,500원 | 컴퓨터 용어사전 / 영진 4,500원 |
| 실무 Lotus 123(제2장) / 영진 12,000원 | 노턴 유틸리티 / 크라운 4,000원 | 컴퓨터 용어사전 / 대은 9,000원 |
| dBASE Ⅱ / 크라운 4,000원 | XENIX V 메뉴얼 / 영진 7,000원 | 미니탭 메뉴얼 / 크라운 6,000원 |
| dBASE Ⅲ / 기전연구사 5,500원 | XENIX 오퍼레이팅 시스템 / 크라운 5,500원 | APPLE Ⅱ 종합분석 / 성안당 5,000원 |
| 실무 dBASE Ⅲ plus / 영진 6,000원 | Dr. HALO Ⅱ / 교학사 3,000원 | APPLE BASIC 프로그램 실습 / 영진 6,000원 |
| 초보자를 위한 dBASE Ⅲ plus / 영진 6,000원 | Dr. HALO Ⅲ / 교학사 3,000원 | 인공지능 / 영진 4,000원 |
| 프로그래머를 위한 dBASE Ⅲ plus / 영진 6,000원 | 센서 인터페이싱 / 기전연구사 24,000원 | 마이크로프로세서 / 동일 7,000원 |

※ 본지에 게재된 기타 신간서적도 주문 가능함

- **주문방법** : 직접 구입 또는 교재명과 출판사명을 기재하여 해당금액에 발송료 500원을 추가하여 우체국 소액환으로 동봉 신청하면 즉시 우송함.
- **보 낼 곳** : ㊖140-012 서울시 용산구 한강로2가 전자단지 17동 3층 가열 325호
  전화 : 716－6024

전문인을 섬기는 **첨 단 서 적**

# 메모리 모델

이용일/한국전자통신연구소

## 메모리모델

MS-DOS상의 대부분의 C 언어처리계는 8086 프로세서의 아키텍처를 반영하여 컴파일러의 옵션에 의해 여러개의 메모리모델을 선택하도록 되어있어 프로그램에 따른 선택을 할 수가 있다. 특히 마이크로소프트 C의 경우에는 언어규격에 독자적인 확장을 하여 하나의 프로그램 내부에서 필요에 따라 near형과 far형을 사용할 수 있고 포인터 및 함수를 하나씩 최적의 명령어로 지정할 수 있다. 이와같이 메모리모델을 혼재시킬 수 있는 것은 보다 고속으로 콤팩트한 프로그램의 기술이 가능하다는 장점이 있는 반면 프로그램 작성자가 관리하는 사항이 증가하기 때문에, 특히 어셈블리 언어와 링크하는 프로그램의 경우에는 인수 및 돌려주는 값이 어떤 메모리모델인가를 파악하지 않으면 프로그램 오류(bug)의 원인이 된다.

여기서는 먼저 마이크로소프트 C의 각 메모리모델에 대해 간략히 설명하고 다음 항에서 어셈블리 언어와 링크하는 프로그램에서 각 데이터의 메모리모델을 어셈블리 언어의 프로그램에서 어떻게 취급해야 하는가를 설명하겠다.

### (1) 스몰모델

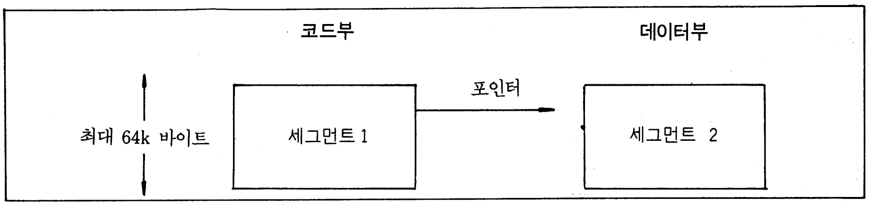
스몰모델(Small Model)은 데이터와 코드를 각각 하나의 세그먼트에 모아 모두를 옵셋(Offset)만으로 액세스할 수 있도록 한 모델이다. 이렇게 하면 포인터를 16비트(near형)로 나타낼 수 있기 때문에, 8086에서도 무리없이 처리할 수 있고 비교적 고속으로 콤팩트한 코드를 생성할 수 있게 된다.

스몰모델에서는 데이터와 코드의 양쪽에 대해 16비트의 포인터(near형)를 이용하기 때문에, 그림 1과 같이 최대로 코드부 64K 바이트와 데이터부 64K 바이트, 합계 128K 바이트 크기의 프로그램을 작성할 수 있게 된다. MS-DOS 상에서 사용하는 대부분의 프로그램은 이 스몰모델로 충분히 기술할 수 있고, 컴파일러의 디폴트(default)도 스몰모델로 되어 있다.

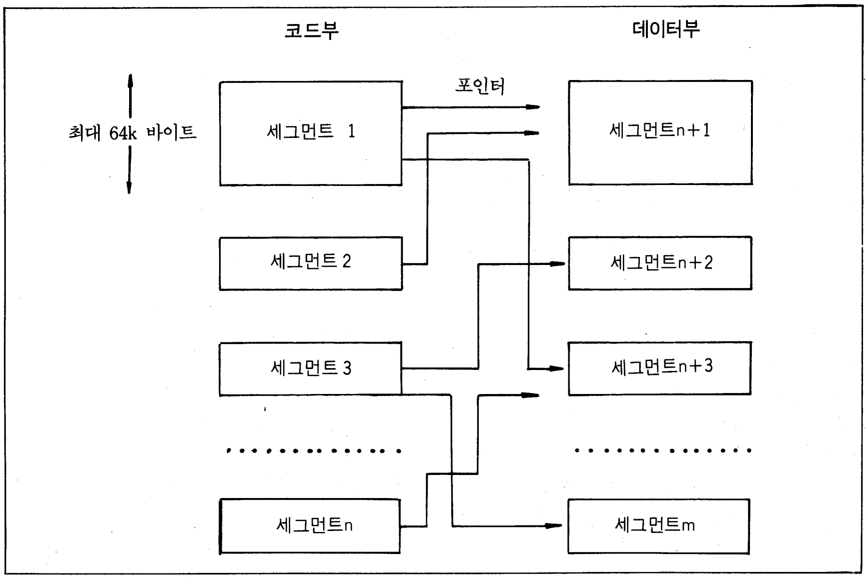
### (2) 라지모델

라지모델(Large Model)은 그림 2와 같이 데이터와 코드를 각각 복수의 세

〔그림 1〕 스몰모델




〔그림 2〕 라지모델



그먼트에 배치하여 항상 세그먼트와 옵셋 모두를 사용하여 액세하도록 한 모델이다. 이 경우, 포인터는 32비트(far형)가 된다. 라지모델에서는 프로그램의 실행속도 및 크기의 면에서 스몰모델보다 불리하지만 프로세서의 한계(실제로는 MS-DOS의 한계)까지의 큰 데이터 및 코드를 취급할 수 있도록 되어 있어, 데이터베이스 시스템 등의 대규모 프로그램을 작성할 수 있게 된다.

### (3) 미디엄모델

스몰모델과 라지모델의 중간적인 모델로서 미디엄모델(Medium Model)이 있다. 미디엄모델에서는 데이터는 near형으로, 코드는 far형(far call)을 이용한다. 따라서 코드의 최대크기는 시스템에서 이용할 수 있는 메모리크기에만 의존할뿐 무제한이고, 데이터는 최대 64K 바이트까지 밖에 사용할 수 없다. **그림 3**에 그 개념을 보였다.

예를 들어 MS-Windows에서 처럼, 어떤 어플리케이션을 동시에 복수개 실행하는 경우에 메모리의 절약을 위해 데이터부 만을 복수개 갖도록 하고 코드부는 공유하게 한다. 이때 데이터부는 고속화를 위해 near형으로 할 수 있지만 코드부는 far형으로 할 수 밖에 없다. 이와 같이 미디엄모델은 코드부분 만을 far형으로 할 필요가 있는 프로그램을 작성할 때 이용한다.

## 데이터의 메모리모델

C언어로부터 호출된 어셈블리 언어의 함수를 기술할 때 그 함수의 인수로서 포인터가 넘겨지는 경우에는 포인터의 메모리모델에 따라 그 인수의 취급이 다르게 된다. C언어 프로그램의 메모리모델은 컴파일 옵션에 의해 쉽게 변경할 수 있는데 반하여 어셈블리 언어의 함수인 경우에는 프로그램의 일부를 명시적으로 변경할 필요가 있기 때문에 조건 어셈블 기능을 이용한다든지 경우에 따라서는 메모리모델마다 별도의 원시 프로그램을 준비하는 등의 대응이 필요하다. 한 예로서, C언어 프로그램에서 어셈블리 언어로 기술한 pcfunc함수를 char형의 포인터 p를 인수로서 호출한 경우를 **프로그램 1**에 보인다. pcfunc의 내부에서는 p의 값을 글로벌한 char형의 포인터 q에 복사하고, C언어의 글로벌변수 c의 주소를 p에 대입한 후에, q의 주소를 돌려주고 있다. 또 뒤에 스몰모델 및 라지모델에 대응하는 어셈블리 언어의 pcfunc함수를 보였지만, 그 전에 pcfunc의 알고리즘을 보이기 위해 C언어로 기술한 경우의 프로그램을 **프로그램 2**에 보인다.

### (1) 포인터가 스몰모델인 경우

포인터가 스몰모델인 경우 세그먼트 부분은 DS 레지스터에 격납되어 있기 때문에 포인터의 값 및 그 실체의 참조, 대입은 옵셋부의 한 워드(Word)만을 다루면 된다. 예를 들면 p의 실체를 액세스하는 경우에는 함수의 인수로서 넘긴 1워드의 옵셋부와 DS 레지스터의 조합으로 액세스할 수 있다. 이 경우의 스택상태를 **그림 4**에 보이고, 스몰모델 용의 pcfunc함수를 **프로그램 3**에 보였다.

〔프로그램 1〕 C언어에서 pcfunc함수를 호출하는 프로그램

```
extern int     *q;
       int     c;

main()
{
        int *p;
        pcfunc(p);
}
```

〔프로그램 2〕 C언어로 작성한 pcfunc함수

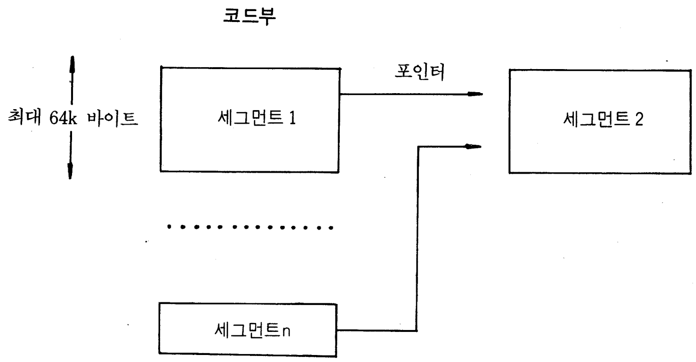
```
/* pcfunc.c */

       int *q;
extern int c;

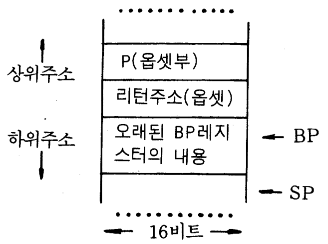
int **pcfunc(p)
int *p;
{
       q = p;
       p = &c;
       return &q;
}

/* ---- end of pcfunc.c ---- */
```

〔그림 3〕 미디엄모델




〔그림 4〕 스몰모델에서 포인터를 넘길때의 스택상태



### (2) 포인터가 라지모델인 경우

라지모델인 경우 포인터는 세그먼트와 함께 넘겨지기 때문에 포인터의 대입 및 데이터의 액세스는 스몰모델과 비교하면 복잡하게 된다. 이 경우의 스택상태를 **그림 5**에, 어셈블리 언어에 의한 프로그램을 **프로그램 4**에 보인다.

어셈블리 언어로 함수를 기술하는 경우에는 가능한 한 라지모델과 스몰모델 양쪽에 모두 대응하는 것이 이상적이지만 그와같은 유연성을 갖게하려면 프로그램을 이해하기 힘들게 될 뿐만 아니라 프로그램의 실행속도면에서도 다소 성능이 떨어지는 경우가 많기 때문에 속도가 생명인 어셈블리 언어의 함수에서는 원시 프로그램을 공유한 경우와 다른 원시 프로그램을 준비한 경우 득실을 따져볼 필요가 있다.

## 함수의 메모리 모델

함수의 메모리 모델은 어셈블리 언어레벨에서는 CALL 명령어로 참조되는 것이 대부분이고 이용방법도 그다지 어렵지 않다. C언어에서 호출된 어셈블리 언어의 함수를 라지모델로 하는 경우에는 PROC 의사명령어로 far형을 선언할 뿐이고 반대로 어셈블리 언어 프로그램에서 C의 함수를 참조하는 경우에는 함수의 선언부에서 그 함수의 형만을 지정하고 이후의 처리는 어셈블리 언어에 맡길 수 있다. 또, 어셈블리 언어의 주소를 포인터로서 취급하는 경우에는 데이터의 경우와 같은 주의가 필요하다.

## 작업용 영역

어셈블리 언어로 기술된 프로그램을 링크하는 경우에는 몇개인가의 세그먼트를 설정해 둘 필요가 있지만 이 세그먼트 이름을 붙이는 방법 및 속성은 마이크로소프트 C와 같게 해두어야 한다. 보통은 코드부는 스몰 모델인 경우 '_TEXT' 혹은 미디엄 및 라지 모델에서는 '〈모듈이름〉_TEXT' 세그먼트에 두고, 데이터부는 '_DATA' 세그먼트에 둔다. 그러나 이것으로는 세

〔프로그램 3〕 스몰모델로 작성한 pcfunc함수

```
;
;       pcfunc.asm (스몰모델용 )  data=small, code=small
;

_TEXT   segment byte public 'CODE'
_TEXT   ends

DGROUP  group   _DATA
        assmume cs:_TEXT, ds:DGROUP

        extrn   _c:word

_DATA   segment word public 'DATA'
        public  _q  ←  q는 외부에서도 참조 가능
_q      dw      0   ←  q가 워드형 변수임을 선언
_DATA   ends

_TEXT   segment byte public 'CODE'

stkframe struc      ┐
bpsave   dw     ?   │ 스택프레임의 구조
retaddr  dw     ?   │
p        dw     ?   │
stkframe ends       ┘

        public  _pcfunc  ←  외부에서 참조가 가능함을 나타냄
_pcfunc proc near ← 함수는 near형으로 호출된다.
        push bp
        mov  bp, sp

        mov  ax, [bp].p  ←  포인터의 값은 워드의 취급과 동일
        mov  _q, ax

        mov  [bp].p, offset DGROUP:_c

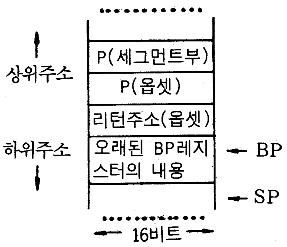
        mov  ax, offset DGROUP:_q ←  함수가 돌리는 값은  ax 레지스터에 격납한다.

        mov  sp, bp
        pop  bp
        ret

_pcfunc endp
_TEXT   ends
        end

; **** end of pcfunc.asm ****
```

〔그림 5〕 라지모델에서 포인터를 넘길때의 스택상태




그먼트가 너무 작은 경우(특히 '_DATA' 세그먼트에 대해서)에는 각 루틴에서 독자적인 세그먼트를 가져도 상관없다. 단, 이때 코드 세그먼트내에 변수영역을 취하게하면 안되고 데이터는 반드시 코드 세그먼트와는 분리하도록 해둔다. 왜냐하면 오버레이(Overlay)의 기능을 저해한다든지 프로그램의 보수가 어렵게 된다든지 하기 때문이다. 또, 앞서 설명한 것처럼 ds, ss, cs의 세그먼트 레지스터의 값을 변경할 경우에는 함수가 종료하기 전에 원래대로 해두는 것을 잊어서는 안된다.

# 실제 함수의 작성

C언어에서 어셈블리 언어를 호출하는 경우의 한 예로서, 펑션 콜을 하기 위한 pcint함수, pcintx함수, segread 함수를 작성하고 이들 함수를 사용하기 위한 인클루드(include) 화일(또는 헤더화일) pcint.h를 설명한다.

이들과 같은 기능의 함수는, MS-DOS 상의 처리계이면 대부분의 C컴파일러에 라이브러리 함수로서 제공되고 있다. 그러나 그들 함수의 규격이 통일되어 있지 않고 또한 그 내부가 공개되고 있지 않는 경우도 많기 때문에 본 연재에서는 여기에서 작성한 함수를 사용하여 펑션 콜을 할 것이다. 또 이들 프로그램은 모두 스몰모델용으로 작성한 것이다. 그러나, 본격적인 소프트웨어 전문자가 아닌 이상은 마이크로소프트사에서 제공하고 있는 각종

〔프로그램 4〕 라지모델로 작성한 pcfunc함수

```
;
;       pcfunc.asm (라지모델용)  data=large, code=small
;


_TEXT   segment byte public 'CODE'
_TEXT   ends

DGROUP  group   _DATA
        assmume cs:_TEXT, ds:DGROUP

        extrn   _c:word

_DATA   segment word public 'DATA'
        public  _q
_q      dd      0 ← 데이터를 라지모델로 했으므로, 포인터도 이중워드(double word)가 된다
_DATA   ends

_TEXT   segment byte public 'CODE'

stkframe struc
bpsave  dw      ?
retaddr dw      ?
p       dd      ?
stkframe ends

        public  _pcfunc
_pcfunc proc near
        push bp
        mov  bp, sp

        les  bx, [bp].p ← LES명령으로 p의 내용을 ES:BX로 읽어  들인다.
        mov  word ptr _q, bx
        mov  word ptr _q+2, es

        mov  word ptr [bp].p, offset DGROUP:_c ┐ 옵셋부 각각을  세그먼트부와  대입
        mov  word ptr [bp].p + 2, _DATA        ┘

        mov  ax, offset DGROUP:_q ┐ 32비트의 데이터를 돌릴때는  DX:AX에 격납한다.
        mov dx, _DATA             ┘

        mov  sp, bp
        pop  bp
        ret

_pcfunc endp
_TEXT   ends
        end

; **** end of pcfunc.asm ****
```

헤더화일 및 라이브러리 그리고 오브젝트 화일들을 이용하더라도 전혀 무리없는 프로그램을 작성할 수 있다. 여기서 작성한 프로그램들은 단지 참고용으로 보존해 두면 좋을 것이다.

## (1) pcint함수

**프로그램 5**에 pcint함수를 보였다. 이 함수는 inregs의 내용을 레지스터에 격납한 후에 INT 21H 즉, 펑션 콜을 수행하는 함수이다. 펑션 콜 종료시의 각 레지스터의 값은 outregs에 격납된다. REGS는 인클루드 화일 pcint.h에서 정의된 공용체이다. 따라서 pcint 함수를 사용하는 경우에는 미리 pcint.h를 인클루드할 필요가 있다.

pcint 함수에서는 세그먼트 레지스터의 값을 지정 또는 취득하는 것은 불가능하기 때문에 그들이 필요한 경우에는 pcintx 함수를 이용해야 한다. 또 REGS의 내용중, 멤버 cflag에는 펑션 콜 종료시의 캐리플래그(Carry Flag)의 내용이 격납되어 있기 때문에 펑션 콜할 때 에러가 발생했든지 어떤지를 알 수 있다.

REGS 공용체 regs는 pcint.h안에 정의되어 있다(**프로그램 6** 참조).

## (2) pcintx 함수

**프로그램 7**의 pcint 함수는 세그먼트 레지스터의 값도 취급할 수 있다는 점을 제외하면 pcint 함수와 같은 기능을 갖는다. sregs는 SREGS형의 구조체이며 pcint.h 안에서 정의되어 있다. pcintx 함수에서는 sregs의 멤버중 ds와 es만이 유효하고 cs와 ss는 무효이다.

sregs는 inregs 및 outregs와 달리 호출할 때의 값과, 돌려주는 값이 변수에 격납되는 것을 주의해야 한다. 또 현재의 세그먼트 레지스터의 내용을 얻기 위해서는 segread 함수를 이용해야 한다. **프로그램 8**에 보인 segread함수는 sregs에 의해 지정된 구조체에 현재의 세그먼트 레지스터의 내용을 격납한다. 이 함수를 이용하는 것에 의해 pcintx 함수를 이용하기 위해 필요한 정보를 줄 수 있다.

sregs는 SREGS형의 구조체이고 인클루드 화일 pcint.h에 정의되고 있다(**프로그램 6** 참조).

이상으로 간략하게나마 메모리 모델에 대해 설명하였다. 다음회에서는 반드시 어떠한 형태로라도 입출력을 동반하게 되는 프로그램을 작성하는 경

〔프로그램 5〕 pcint함수

```
;         *****    pcint.asm     *****
regs      struc        ┐
rax       dw      ?    │
rbx       dw      ?    │
rcx       dw      ?    │  인수의 구조체 형 선언
rdx       dw      ?    │
rsi       dw      ?    │
rdi       dw      ?    │
cflag     dw      ?    │
regs      ends         ┘

stkframe struc         ┐
bpsave    dw      ?    │
retaddr   dw      ?    │  _pcfunc의 스택프레임 선언
inregs    dw      ?    │
outregs   dw      ?    │
stkframe ends          ┘

_TEXT     segment byte public 'CODE'  ← 마이크로소프트 C인 경우
          assume  cs:_TEXT
          public  _pcint
_pcint    proc near  ← 라지모델인 경우는 far선언
          push bp      ┐
          mov  bp, sp  ┘ 스택프레임 작성
          push di ┐
          push si ┘ DI와 SI 레지스터는, 마이크로소프트 C의 제약
                    때문에 보존할 필요가 있다.
          mov  di, [bp].inregs
          mov  ax, [di].rax  ┐
          mov  bx, [di].rbx  │
          mov  cx, [di].rcx  │ 레지스터 값의 설정
          mov  dx, [di].rdx  │   (← inregs)
          mov  si, [di].rsi  │
          mov  di, [di].rdi  ┘
          int 21h  ← 펑션 콜
          push di               ┐
          mov  di, [bp].outregs │
          mov  [di].rax, ax     │ 레지스터 값의 격납
          mov  [di].rbx, bx     │   (→ outregs)
          mov  [di].rcx, cx     │
          mov  [di].rdx, dx     │
          mov  [di].rsi, si     │
          pop  [di].rdi         ┘
          mov  si, 1
          jc   carry
          xor  si, si
carry:    mov  [di].cflag, si
          pop  si
          pop  di
          pop  bp
          ret
_pcint    endp
_TEXT     ends
          end
```

우에 필수불가결한 디바이스 및 필터와의 입출력에 대한 지식에 대하여 설명하고 MS-DOS의 제닉스 호환 펑션콜을 중심으로 하여 펑션콜을 할 때 어떻게 여러가지 입출력을 실현하는가 설명할 것이다.

〔프로그램 6〕 pcint, pcintx함수의 헤더

```
/* word registers : 16 비트 레지스터 */

struct WORDREGS {
        unsigned int ax;
        unsigned int bx;
        unsigned int cx;
        unsigned int dx;
        unsigned int si;
        unsigned int di;
        unsigned int cflag;
        };

/* byte registers : 8 비트 레지스터 */

struct BYTEREGS {
        unsigned char al, ah;
        unsigned char bl, bh;
        unsigned char cl, ch;
        unsigned char dl, dh;
        };

/* general purpose registers union - overlays the corresponding
 * word and byte registers.
*/

/* 레지스터를 나타내는 공용체 (union) */

union REGS {
        struct WORDREGS x;
        struct BYTEREGS h;
        };

/* 세그먼트 레지스터 */

struct SREGS {
        unsigned int es;
        unsigned int cs;
        unsigned int ss;
        unsigned int ds;
        };

int pcint();
int pcintx();
void segread();

/* end of pcint.h */
```

〔프로그램 7〕 pcintx함수

```
;       *****   pcintx.asm   *****   ←
regs    struc
rax     dw      ?       DS, ES 레지스터의 값도 넘길 수
rbx     dw      ?       있는 pcint 함수의 확장 프로그램
rcx     dw      ?
rdx     dw      ?
rsi     dw      ?
rdi     dw      ?
cflag   dw      ?
regs    ends

sreg    struc
res     dw      ?
rcs     dw      ?
rss     dw      ?
rds     dw      ?
sreg    ends

stkframe struc
bpsave  dw      ?
retaddr dw      ?
inregs  dw      ?
outregs dw      ?
sregs   dw      ?
stkframe ends

_DATA   segment word public 'DATA'
_DATA   ends

DGROUP  group   _DATA

_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT,  ds:DGROUP

        public  _pcintx
_pcintx proc near
        push bp
        mov  bp, sp
```

```
        push di
        push si
        push ds
        push es

        mov  di, [bp].inregs
        mov  ax, [di].rax
        mov  bx, [di].rbx
        mov  cx, [di].rcx
        mov  dx, [di].rdx
        mov  si, [di].rsi
        push [di].rdi
        mov  di, [bp].sregs
        mov  es, [di].res
        mov  ds, [di].rds
        pop  di

        int 21h

        push di
        push ds ──────────┐
        mov  di, DGROUP    │ DS 레지스터의 값은 일단 스택에
        mov  ds, di        │ 대피하고, 세그먼트 주소를 얻은
        mov  di, [bp].sregs│ 이후에 sregs에 격납한다.
        mov  [di].res, es  │
        pop  [di].rds ←───┘
        mov  di, [bp].outregs
        mov  [di].rax, ax
        mov  [di].rbx, bx
        mov  [di].rcx, cx
        mov  [di].rdx, dx
        mov  [di].rsi, si
        pop  [di].rdi
        mov  si, 1
        jc   carry
        xor  si, si
carry:  mov  [di].cflag, si
        pop  es
        pop  ds
        pop  si
        pop  di
        pop  bp
        ret
_pcintx endp
_TEXT   ends
        end
```

〔프로그램 8〕 Segread함수

```
;
;       *****   segread.asm    *****
;
sreg    struc   ┐
res     dw    ? │ pcint.h 에 선언되어 있는 구조체 sregs와
rcs     dw    ? │ 같은 것
rss     dw    ? │
rds     dw    ? │
sreg    ends    ┘

stkframe struc
bpsave  dw    ?
retaddr dw    ?
sregs   dw    ?
stkframe ends

_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT

        public  _segread
_segread proc near
        push bp              ┐
        mov  bp, sp          │ 스택프레임의 생성
        mov  bx, [bp].sregs  ┘

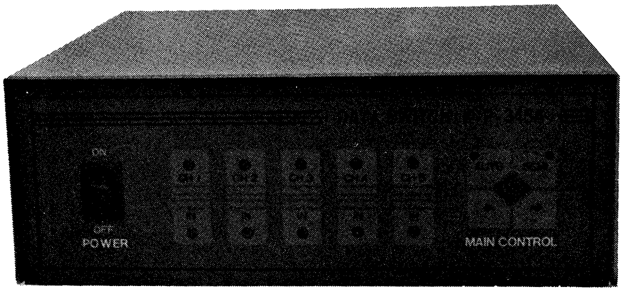
        mov  [bx].res, es    ┐
        mov  [bx].rcs, cs    │ 세그먼트 주소를 읽어 들인다
        mov  [bx].rss, ss    │
        mov  [bx].rds, ds    ┘
        pop  bp
        ret
_segread endp

_TEXT   ends
        end
```

# PRINTER 주변기기 전문 메이커

- 컴퓨터는 2대 혹은 3대이고, 프린터는 1대뿐이라 불편하게 사용하고 계십니까?
- 컴퓨터 1대에 프린터를 1대씩 사용하거나 추가 구입시 프린터를 구매하려고 생각하십니까?
- 프린터를 하고 있는 동안 컴퓨터를 장시간 사용하지 못하고 계십니까?
- CAD 사용시 Plotter가 그리고 있는 동안 컴퓨터를 장시간 사용하지 못하고 계십니까?
- 컴퓨터의 수량과 프린터의 수량이 안맞아 적절한 사용을 못하고 계십니까?
- 학교 및 학원의 실습실에 프린터대수가 부족해 적절한 교육을 못하고 계십니까?
- F.A 혹은 계측기쪽의 프린터 인터페이스에 문제가 있읍니까?

## 컴퓨터 사용중에 불편한 위의 모든 문제를 자신있게 처리해 드리므로, 귀사의 업무에 도움이 되어 드리고 싶읍니다!



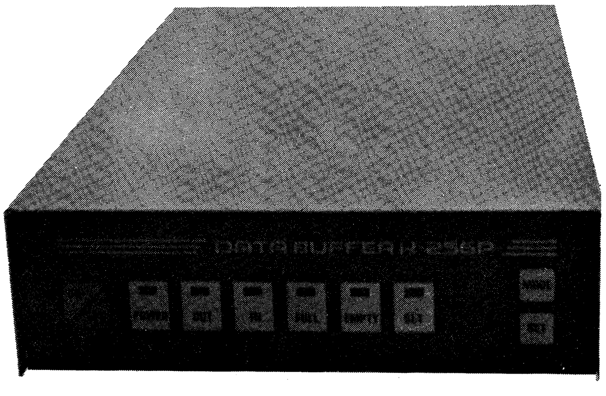

※ K-P-345A는 컴퓨터 프로그램 보호법에 의해 등록된 제품입니다.

**K-P-345A의 사양**

| | |
|---|---|
| 입·출력 방식 | Centronics Type |
| 입력 COMPUTER | 3 : 1~5 : 1 |
| 출력 PRINTER | 1 대 |
| 대기 시간 | 5초, 10초, 20초, 40초 중 선택 |
| 채널 변환방식 | 전자식 |
| 채널 선택방식 | 자동 or 반자동 |
| 용지 F.F 기능 | 자동 F.F or None F.F |
| F.F 삭제 기능 | 자동 삭제 or None |

**K-256P의 사양**

| | |
|---|---|
| 버퍼 용량 | 256kB(80col 약 200장) |
| 입·출력 방식 | Centronics Type |
| 입·출력 수 | 1 : 1 or 2 : 1 |
| 사용자 MODE | 10EA |
| 고유 기능 | Data 중 Space 입력 기능, 재 Copy 기능, 메모리 기능, Self Test 기능 등 총 10가지 |




## DATA SWITCH

- 전자식 완전자동 (2 : 1~8 : 1 or 64 : 1)
- 전자식 반자동 (2 : 1~8 : 1, 4 : 2)
- Serial 완전자동(4 : 1, 8 : 1)

## DATA BUFFER

- 소용량 64kB
- 고성능 다기능 256kB
- 대용량 다기능 1MB
- Serial용 1MB

## DATA CONVERTOR

- Serial→PARALLEL
- PARALLEL→Serial

## 각급 학교 및 학원의 전산 실습실 프린터 공유장치 공사

**ATTA SYSTEM**
PRINTER 주변기기 전문메이커
서울 · 종로구 종로 3가 175-4 (현대APT1309호)
전 화 : 2 7 7 - 1 7 3 9
F A X : 2 7 7 - 8 2 5 1

● 상기 제품은 아래의 전문 취급점에서 전시 판매하고 있으며 직접 사용해 보실 수 있읍니다.●

**서울** : 세운상가 협진 274—5419(가동 라열 426호)
용산 나진상가 동해전자 719—8010(17동 가열 304호)
용산 전자랜드 협진 712—2764(3층 C-336호)

**인천** : 바이너 리컴퓨터 (032)762-5544
**대구** : 범진 컴퓨터 (053)252-2211
**마산** : 한길 컴퓨터 (0551)94-5555
**원주** : 중부 시스템 (0371)43-7979
(기타 지역 전문 취급점 모집중)

# 플래그 레지스터를 읽기 쉽게

김한승/고려컴퓨터학원장

각종 CPU의 기계어를 다루다 보면 여러 종류의 레지스터를 접하게 된다. 그중 독특한 것 중의 하나가 플래그 레지스터(Flag Register)인데 다른 레지스터들은 여러 비트들이 모여서 하나의 워드를 구성하고 있는데 비해 이 플래그 레지스터의 비트들은 각각 독립적으로 분산되어 CPU의 연산결과 등에 관한 현재의 상태를 알려주는 기능을 갖고 있다.

물론 고급언어를 이용하여 프로그래밍할 경우에는 구태여 레지스터들을 의식할 필요가 없지만 기계어 및 어셈블리 언어를 이용할 때는 꼭 알아야 하는 것들이다. 특히 플래그 레지스터는 CPU의 상태를 알려주는 일종의 '신호등' 역할을 하고 있으므로 그 중요성은 새삼 강조할 필요가 없을 줄로 안다.

실제로 우리 주변을 보면 기계어를 다루는 사람은 거의 없고 주로 어셈블리 언어를 쓰는 사람이 많음을 볼 수 있다. 앞에서도 언급했듯이 이 언어는 로우 레벨(Low Level)언어이기 때문에 CPU의 상태를 프로그래머가 직접 감지하면서 프로그램을 작성해야 하는데 IBM-PC에서는 이를 위하여 DEBUG라는 유용한 유틸리티를 제공하고 있다. (비단 이 DEBUG뿐만 아니라 다른 디버깅 유틸리티가 있더라도 CPU와 레지스터의 상태를 알려주는 기능은 기본적으로 갖추고 있다.)

대부분의 DOS디스켓 속에 들어 있는 DEBUG유틸리티는 앞에서 설명한 플래그의 상태를 2개의 약자로 표현하고 있다. 그런데 이 약자표현은 영어문화권의 사용자들을 위주로 한 것으로 한글을 쓰는 우리에게는 그리 쉽게 와닿지 않는게 사실이다. 한 예로 오버플로우 플래그인 경우 'NV', 'OV'로 상태를 표현하고 있는데 한글문화권의 사용자는 얼마후에야 'NV'는 'Not oVerflow', 'OV'는 'OVerfolw'임을 생각하게 된다. 더구나 8088 CPU에는 9개의 플래그가 쓰이고 있기 때문에 플래그를 항시 참조해야 하는 어셈블리 언어 사용자들이 이 9개의 약자를 참조하는 일은 여간 신경쓰이는 일이 아닐 수 없다. (어셈블리 언어를 사용하고 있는 독자들은 쉽게 공감하리라 본다.) 그래서 필자는 간단한 방법으로 플래그의 상태를 읽기쉽게 개조해서 쓰고 있는데 독자들에게도 도움이 되리라 믿고 소개하는 것이다. 알고 보면 간단한 것인데 이 생각을 미처 못하고 아홉개의 약어를 외우는 것은 시간의 낭비는 물론 혼동만 가져올 뿐이다.

## 플래그 레지스터

플래그(Flag)는 말 그대로 '깃발'이란 의미인데 깃발이 어떤 상태를 나타내는데 쓰이듯이 플래그도 CPU의 상태를 나타내 주는 기능을 갖고 있다. 즉 8088 CPU속에 1비트로 구성되어 있는 레지스터로 이 1비트 레지스터에 0이 세트되어 있는가 1이 세트되어 있는가에 따라 특정 상태를 나타내게 된다. 이들 플래그는 '플래그 레지스터'라는 16비트 레지스터 내에 존재하고 있는데 실제 배치상태는 **그림 1**에 나타낸 바와 같다.

먼저 각 플래그들의 기능을 간단히 살펴 보기로 하자.

- **캐리 플래그(CF)**: 자리올림(Carry) 혹은 최상의 비트로부터 빌림(Borrow)이 발생하면 1로 세트되고 그 이외의 경우는 0으로 리세트 된다.
- **패리티 플래그(PF)**: 연산한 결과 1로 된 비트의 수가 짝수개(Even number)일 때 1로 세트되고 홀수(Odd number)일 때 0으로 리세트된다.
- **보조 캐리 플래그(AF)**: 8(16)비트 연산에서 하위 4(8)비트로부터 상위 4(8)비트로 자리올림 혹은 빌림이 발생한 경우에 1로 세트되고 그 이외의 경우 0으로 리세트된다.
- **제로 플래그(ZF)**: 연산한 결과가 0으로 되었을 경우에 1로 세트되고 그 이외의 경우에 0으로 리세트된다.
- **사인 플래그(SF)**: 연산한 결과, 최상위 비트가 1이 되었을 때(즉 보수 표현으로 음수가 되었을 때) 1로 세트되고 그 이외에는 0으로 리세트된다.
- **오버플로우 플래그(OF)**: 연산을 부호가 달린 숫자로 했을 때 오버플로우, 혹은 언더플로우가 발생한 경우에 1로 세트되고 그 이외에는 0으로 리세트된다.

이상의 6가지 플래그는 연산의 결과에 따라서 세트 혹은 리세트되는 플래그로써 상태 플래그(Status Flag)라 부른다. 다음의 3가지 플래그는 CPU의 동작 상태를 제어하는 것으로 제어 플래그(Control flag)이다.

- **디렉션 플래그(DF)**: 스트링 조작을 할 때는 이 플래그가 0이면 번지를 나타내는 레지스터 값이 자동적으로 증가하고 1이면 레지스터 값이 자동적으로 감소한다.

- 인터럽트 플래그(IF) : 이 플래그가 0일때 INTR 단자로부터의 외부 인터럽트 요구는 무시되고 1일때는 외부 인터럽트 요구를 받아 들일 수 있게 된다.
- 트랩 플래그(TF) : 이 플래그가 0일때 CPU는 보통대로 명령을 실행한다. 1일때는 CPU가 하나의 명령을 실행할 때마다 자동적으로 내부 인터럽트(INT 1)를 발생시키고 인터럽트 처리루틴으로 들어 간다.(주로 프로그램의 추적에 이용됨)

표 1은 이들 플래그의 상태를 DEBUG유틸리티에서 표현하는 약자와 의미를 보인 것이다.

## 수정방법

이 글을 읽고 난 후에는 독자들의 취향에 따라 여러 형태로 수정할 수 있으나 일단 여기서는 비트가 'ON'일때, 플래그와 이름 1자에 '*'를 표시하는 것으로 한다.

DEBUG의 수정작업은 DEBUG자신의 프로그램으로 하는데 DEBUG의 확장명은 주로 '.COM'으로 되어 있지만 간혹 '.EXE'로 된 것도 있다. 확장자명이 EXE인 경우은 디스크에 완전한 실행코드 형태를 갖추지 않은 상태였다가 메모리로 읽어 들일때 재할당(Relocation) 정보의 도움을 받아 로드되고 있다. 즉 메모리로 읽어들인 후에는 디스크에 저장된 내용과 달라지게 되며 메모리에서 디스크로 저장시에는 그대로 저장된다. 그래서 DEBUG에서는 EXE의 확장명을 갖는 화일은 Write(W)명령이 안돋도록 되어 있다.

일단 확장명이 'EXE'이든 'COM'이든 작업을 쉽게 하기위해 아무 확장자도 가지지 않는 화일명으로 DEBUG프로그램을 하나 만든다.

```
A:\>COPY DEBUG.COM DEBUG ←—— 확장명이 없는
                             DEBUG화일로 복사한다.
```

다음은 기존의 DEBUG를 실행시키면서 수정용 DEBUG를 읽는다.

```
A:\>DEBUG DEBUG
      ↑     ↑
  실행화일  작업화일
```

DEBUG의 프롬프트(-)가 나오면 플래그 상태의 표시문자를 찾기 위해 다음 같이 입력한다.

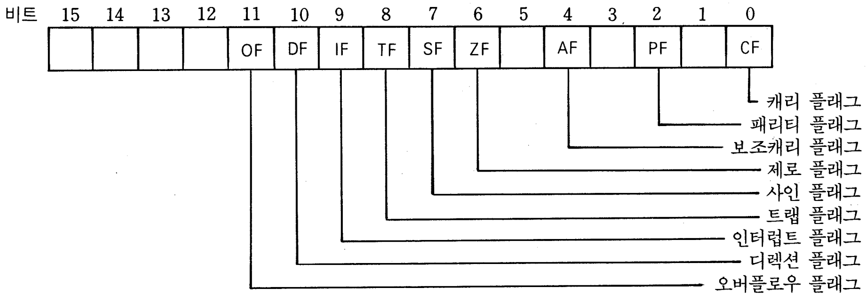
```
-S0 4000 'OVDNEI'
1045:358F
-D3580
```

여기서 'S'명령은 오프셋 번지 0에서 400H 사이에서 뒤의 문자열(대문자) 'OVDNEI'를 찾는 서치(Search)명령이다(대개의 버전에서 DEBUG의 길이는 4000H바이트 이하임).

화면 1은 위의 명령을 실행시킨 결과인데 여기서 보면 각 플래그표시 문자가 ON, OFF별로 2자씩 정의되어 있음을 볼 수 있다. 문제는 이들 문자를 원하는 형태로 고쳐주기만 하면 해결되는 것으로 다음과 같이 입력한다.

```
-a358f
1045:358F db 'O*D*I*',0,0,'S*Z*',
    0,0,'A*',0,0,'P*',0,0,'C*'
1045:35A7
-a35af
1045:35AF db 'O D I ',0,0,'S Z ',
    0,0,'A ',0,0,'P ',0,0,'C '
```

(그림 1) 플래그 레지스터의 배열상태




(공백인 비트는 사용하지 않음)

(표 1) DEBUG.COM에서 사용되는 약자

| 상태 \ 플래그 | OF | DF | IF | SF | ZF | AF | PF | CF |
|---|---|---|---|---|---|---|---|---|
| 0 | NV | UP | DI | PL | NZ | NA | PO | NC |
| 1 | OV | DN | EI | NG | ZR | AC | PE | CY |

(a) DEBUG.COM에서 쓰이는 약자

| 플래그 \ 상태 | 0 | 1 |
|---|---|---|
| 오버플로우 플래그 | Not oVerflow | OVerflow |
| 디렉션 플래그 | UP | DowN |
| 인터럽트 플래그 | Disable Interrupt | Enable Interrupt |
| 사인 플래그 | PLus | NeGative |
| 제로 플래그 | Not Zero | ZeRo |
| 보조캐리 플래그 | Not Auxiliary carry | Auxiliary Carry |
| 패리티 플래그 | Parity Odd | Parity Even |
| 캐리 플래그 | Not Carry | CarrY |

(b) 각 약자의 의미

```
1045:35C7
```

여기서 '0,0'로 된 부분은 사용하지 않는 플래그로 반드시 '0'으로 채워야 하며 정확한 번지에 정확한 문자가 기억되었는지 확인하는 것을 잊지 말자(화면 2 참조).

이상이 없으면 수정된 화일을 디스크에 저장시킨 후 DOS로 빠져 나온다.

```
-W
Writing 3E19 bytes
-Q
A:\>
```

마지막으로 기존의 DEBUG.COM을 지워 버리고 수정된 DEBUG를 DEBUG.COM으로 이름을 고친다.

```
A:\>DEL DEBUG.COM
A:\>REN DEBUG DEBUG.COM
A:\>
```

이제는 모든 작업이 끝났다. 이제부터 이 DEBUG는 플래그 비트가 ON이면 '*'표시를 나타내 주므로 플래그의 상태를 쉽게 알 수 있을 것이다(화면 3 참조).

〔화면 1〕 플래그 상태를 나타내는 표시문자

```
A:\>
A:\>DEBUG DEBUG
-S0 4000 'OVDNEI'
1045:358F ←——— 찾은 번지를 DEBUG가 알려줌(세그먼트 번지 1045는 다를 수 있음)
-D3580 ←——— 찾은 부분을 덤프시킨다.
1045:3580  00 49 50 00 50 43 00 00-00 00 00 00 00 00 00 4F   .IP.PC.........O
1045:3590  56 44 4E 45 49 00 00 4E-47 5A 52 00 00 41 43 00   VDNEI..NGZR..AC.   플래그 ON
1045:35A0  00 50 45 00 00 43 59 00-00 00 00 00 00 00 00 4E   .PE..CY........N
1045:35B0  56 55 50 44 49 00 00 50-4C 4E 5A 00 00 4E 41 00   VUPDI..PLNZ..NA.   플래그 OFF
1045:35C0  00 50 4F 00 00 4E 43 00-00 00 00 00 00 00 00 00   .PO..NC.........
1045:35D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1045:35E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1045:35F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

〔화면 2〕 수정된 후의 표시문자

```
 D3580
1045:3580  00 49 50 00 50 43 00 00-00 00 00 00 00 00 00 4F   .IP.PC.........O
1045:3590  2A 44 2A 49 2A 00 00 53-2A 5A 2A 00 00 41 2A 00   *D*I*..S*Z*..A*.   플래그 ON
1045:35A0  00 50 2A 00 00 43 2A 00-00 00 00 00 00 00 00 4F   .P*..C*........O
1045:35B0  20 44 20 49 20 00 00 53-20 5A 20 00 00 41 20 00    D I ..S Z ..A .   플래그 OFF
1045:35C0  00 50 20 00 00 43 20 00-00 00 00 00 00 00 00 00   .P ..C .........
1045:35D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1045:35E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1045:35F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-W
Writing 3E19 bytes
-Q
```

〔화면 3〕 이제는 플래그의 상태를 쉽게 알 수 있다.

```
A:\>DEL DEBUG.COM ——— 기존의 DEBUG를 삭제한다.

A:\>REN DEBUG DEBUG.COM ——— 수정된 DEBUG를 실행화일로 교체한다.

A:\>
A:\>DEBUG
-T

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1045  ES=1045  SS=1045  CS=1045  IP=3E87   O  D  I* S  Z  A  P  C
1045:3E87 E80000        CALL    3E8A
-
-T

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEC  BP=0000  SI=0000  DI=0000
DS=1045  ES=1045  SS=1045  CS=1045  IP=3E8A   O  D  I* S  Z  A  P  C
1045:3E8A 5B            POP     BX
-T
```

# LIFEBOAT

The software publisher

| 제 품 명 | 가 격 |
|---|---|
| **언어/Assemblers** | |
| 386 ASM Link & Locate | 690,900 |
| Macro Assembler(MASM) | 128,000 |
| Turbo Assembler/Debugger | 138,500 |
| **BASIC** | |
| QuickBasic | 90,200 |
| True Basic | 116,300 |
| **C** | |
| C Compiler(Lattice) | 325,200 |
| C Compiler(Microsoft) | 373,600 |
| C Optimizing Compiler | 312,200 |
| C++ Compiler | 156,800 |
| High C-286 | 777,100 |
| High C-386 | 1,142,800 |
| Instant C/16M | 1,004,400 |
| NDP C | 717,000 |
| Quick C | 90,200 |
| Turbo C | 138,500 |
| Turbo C Professional | 229,900 |
| **COBOL** | |
| COBOL | 822,800 |
| Realia COBOL | 1,121,900 |
| RM/Cobol-85 | 1,287,800 |
| **dBASE** | |
| Clipper | 560,300 |
| dBASE IV | 710,500 |
| dBMAN V | 194,600 |
| dBXL Diamond | 173,700 |
| Foxbase+ 386 | 495,000 |
| Foxbase+ Dev Pkg SU | 308,300 |
| Quicksilver Diamond | 521,100 |
| **DBMS** | |
| Advanced Revelation | 834,600 |
| Clarion Finance 1 LEM | 39,200 |
| Clarion Personal Dev | 146,300 |
| Clarion Professional Dev | 625,600 |
| D the data language | 301,700 |
| Dataflex MU | 1,317,800 |
| Emerald Bay Eagle | 450,600 |
| Magic PC | 245,600 |
| Paradox 2.0 | 690,900 |
| R:BASE for DOS | 717,000 |
| **Expert Systems** | |
| Exsys Professional | 696,100 |
| Personal Consultant Plus | 2,663,000 |
| **FORTRAN** | |
| FORTRAN Opt Compiler | 416,700 |
| FORTRAN-F77L | 610,000 |
| NDP FORTRAN-386 | 717,000 |
| Personal FORTRAN | 116,300 |
| **LISP** | |
| PC Scheme | 90,200 |
| TransLISP PLUS w/Runtime | 403,600 |
| **Modula·2** | |
| Modula-2 Dev. System | 233,800 |
| Topspeed Modula-2 | 120,200 |
| **Object Oriented** | |
| ACTOR | 538,100 |
| Smalltalk/V | 105,800 |
| Smalltalk/V 286 | 210,300 |
| **Other Languages** | |
| RPG II Complete | 1,640,400 |
| Trilogy | 122,800 |
| **PASCAL** | |
| Turbo Pascal | 138,500 |
| Turbo Pascal Prof | 229,900 |
| **PROLOG** | |
| Arity Combo Pack | 1,154,600 |
| SQL Development Pack | 312,200 |
| Turbo Prolog | 134,600 |
| Turbo Prolog Toolbox | 90,200 |
| **라이브러리/AI & Object Oriented** | |
| C+ Objects Vol 1 | 560,300 |
| CxPERT | 508,100 |
| **Communications** | |
| AdComm | 295,200 |
| C ASYNC MANAGER | 151,500 |
| Comm Library | 233,800 |
| Comm Pak w/Brkout | 259,900 |
| Multi Comm | 129,300 |
| NET-TOOLS | 129,300 |
| SilverComm | 160,700 |
| Turbo ASYNCH PLUS | 113,700 |
| **File Managements** | |
| AXOS | 482,000 |
| Btrieve | 534,200 |
| C index+ | 346,100 |
| c-tree | 347,400 |
| CBTREE | 134,600 |
| db_FILE-SU-OBJ | 390,500 |
| d-tree | 436,300 |
| db2c Files | 86,200 |
| dbc db3 | 190,700 |
| dbcPlus Source Only | 782,300 |
| WKS Library | 193,300 |
| **General** | |
| "C"erious Toolkit Plus | 117,600 |
| Blackstar C | 113,700 |
| C Tools Plus | 113,700 |
| C Utility | 175,100 |
| Clipper Toolbox | 325,200 |
| dBASE Program Util | 78,400 |
| dBASE Program Util Vol 2 | 78,400 |
| dBASE Tools for C | 78,400 |
| Functions | 220,800 |
| QuickPak Professional | 129,300 |
| Spindrift Lib | 146,300 |
| SuperFunctions | 273,000 |
| Tom Retting Library | 90,200 |
| Turbo Plus | 105,800 |
| Turbo Power Tools Plus | 113,700 |
| **Graphics** | |
| dBASE Tools for C Graphic | 78,400 |
| dGE | 190,700 |
| Essential Graphics | 269,100 |
| Flipper | 177,700 |
| GraphiC | 403,600 |
| GraphPak | 60,100 |
| GSS Graphics Dev Tlkt | 521,100 |
| GSS Kernel | 521,100 |
| HALO-88 | 297,800 |
| HOOPS 32 | 608,600 |
| Laser Pak Professional | 129,300 |
| Metawindows | 176,400 |
| PCX Programmer's Toolkit | 129,300 |
| **Math** | |
| Business Mathlib | 333,100 |
| **OS Enhancements** | |
| C Sharp | 495,000 |
| cINT | 116,300 |
| DESQview API C Lib/Ref | 176,400 |
| Multi C | 129,300 |
| OS/2 Programmer's Toolkit | 325,200 |
| Resident C w/source | 173,700 |
| Security Lib w/source | 283,500 |
| Wendin PC UNIX | 101,900 |
| **Query** | |
| CBTREE/SQL | 360,500 |
| CQL Full w/Source | 351,400 |
| dQUERY/Lib | 337,000 |
| db_RETRIEVE-SU-OBJ | 390,500 |
| XQL | 717,000 |
| Xtrieve | 220,800 |
| **Report** | |
| CB/IQ Report Dev Version | 273,000 |
| r-tree | 258,600 |
| Report Option | 141,100 |
| R & R Rel Reportwriter | 134,600 |
| **Sort** | |
| Opt-Tech Sort | 142,400 |
| **Text Screen** | |
| C Display Manager | 120,200 |
| C-Scape | 390,500 |
| C-Worthy | 220,800 |
| C-Worthy w/Forms | 338,300 |
| Curses | 193,300 |
| DataWindow | 299,100 |
| dBrief Int'l | 103,200 |
| HI-SCREEN XL | 129,300 |
| I/O Pro | 155,500 |
| MakeForm | 125,400 |
| Multi Windows | 259,900 |
| PANEL PLUS | 429,700 |
| POWER SCREEN | 113,700 |
| ScreenStar w/source | 117,700 |
| SoftCode | 75,800 |
| UI Programmer | 246,900 |
| VC Screen | 154,200 |
| Vitamin C | 198,600 |
| Windows for Data w/maint | 1,409,200 |
| **툴/Debuggers & Disassemblers** | |
| 386 Debug | 181,600 |
| Advantage Disassembler | 351,400 |
| dbug for dbase | 181,600 |
| DIS*DOC | 155,500 |
| Periscope II | 151,500 |
| Periscope III | 1,500,600 |
| Soft Probe II/TX | 377,500 |
| Soft-X-Plore | 163,300 |
| Sourcer w/BIOS pre-proc | 155,500 |
| **Diagrammers & Documentors** | |
| ASM Flow | 99,300 |
| Clear for dBASE | 116,300 |
| Design/2.0 | 455,800 |
| FlowCharting II Plus | 207,700 |
| FoxToolBox | 351,400 |
| Interactive EasyFlow | 176,400 |
| Source Print | 101,900 |
| Turbo Analyst | 83,600 |
| **Editors** | |
| Brief Int'l w/Macro | 203,800 |
| Edix | 155,500 |
| Epsilon | 181,600 |
| KEDIT | 155,500 |
| MKS VI | 155,500 |
| Personal Rexx | 155,500 |
| PI Editor | 246,900 |
| SPF/PC | 224,700 |
| VEdit+ MM | 168,500 |
| **Help Guides & Training** | |
| C Trainer | 134,600 |
| DBASE ON LINE | 100,600 |
| Norton Guides for OS/2 | 120,200 |
| Visible Computer:8088 | 73,200 |
| Visible Computer:80286 | 87,600 |
| **Linkers & Libraries** | |
| Link & Locate ++ | 376,200 |
| PLINK-86 Plus | 377,500 |
| PolyLibrarian II | 168,500 |
| **Make Utilities** | |
| PolyMake | 168,500 |
| **Profilers & Generators & CASE** | |
| Inside! | 75,800 |
| PC Metric | 181,600 |
| Stopwatch | 64,000 |
| **Prototypers, Generators & CASE** | |
| COMPEDITOR | 185,500 |
| Dan Bricklin Demo II | 163,300 |
| Genifer | 312,200 |
| Instant Replay III | 158,100 |
| Matrix Layout | 129,300 |
| Proteus | 90,200 |
| Show Partner F/X | 368,300 |
| Sycero db Net | 795,400 |
| **Lints** | |
| PC Lint | 116,300 |

# 꼭 필요하여 애써 찾으시던, 그러나 어디서 어떻게 구입할 지 몰랐던 최신 버젼 PC용 시스템소프트웨어●유틸리티●툴키트● 응용패키지들을 바로 여기서 구입하십시요.

| | |
|---|---|
| **Text & UNIX Tools** | |
| C Toolset Int'l | 107,100 |
| MKS AWK | 95,400 |
| PolyAWK | 112,400 |
| Turbo Professional | 115,000 |
| **Translators** | |
| BAS_C Commercial | 350,100 |
| Compile 1-2-C | 291,300 |
| db2c Toolkit | 259,900 |
| FOR_C | 495,000 |
| **Version Control** | |
| MKS RCS | 196,000 |
| PVCS Corporate | 450,600 |
| **응용프로그램/Backup & Recovery** | |
| Norton Utilities Advanced | 120,200 |
| PC Tools Deluxe | 69,300 |
| **Communications & Terminal Emulator** | |
| Co/Session 2 User | 220,800 |
| Sidetalk | 101,900 |
| **DTP & Text Formatting** | |
| Poly Doc SU | 220,800 |
| RUNOFF | 53,600 |
| **OS Enhancements** | |
| 386 to the MAX | 74,500 |
| DESQview | 112,400 |
| MKS Trilogy | 116,300 |
| Norton Commander | 69,300 |
| PolyShell | 112,400 |
| Soft Bytes 386 Max | 77,100 |
| VM-386 | 243,000 |
| Windows/386 | 177,700 |
| **Other OS** | |
| QEMM-386 | 58,800 |
| **Security** | |
| Virusafe | 159,400 |
| Watch Dog | 312,200 |
| **TSR's & DOS Utilities** | |
| Auto Might | 60,100 |
| FTOOLS | 90,200 |
| **DOS 외 OS /Macintosh** | |
| Aztec C68K-c | 521,100 |
| **UNIX & XENIX** | |
| Btrieve/Xenix | 534,200 |
| db_RETRIEVE-MU sou only | 1,383,100 |
| DOSMerge 386 limited | 495,000 |
| DOSMerge 386 unlimited | 638,700 |
| Edix-SU | 290,000 |
| NDP-C 386 | 717,000 |
| SysV/286 Complete unltd | 1,076,200 |
| SysV/386 Complete unltd | 1,357,000 |
| Xenix Complete System 286 | 1,383,100 |

### VITAMIN C

Veteran C programmers have known the unmatched performance of VITAMIN C for years. Now, everyone is finding that VITAMIN C-the pioneer in data windows-is THE user interface. High level functions mean the fast track to better applications.

VITAMIN C saves you time and money with

- Windows-Multiple & Bullet-proof
- Windows-Overlapping & Pop-up
- Windows-Zoom & Move & Scroll
- Windows-Hide & Access ANY TIME
- Data Entry-Formatting & Validation
- Help-Context Sensitive
- Menus-Pop-up & Pull-down & Multi-level
- Keyboard Handler & Idle Fuction
- Text editor & Printer Output Functions
- VCScreen option for interactive code generation

### TVC 80286

A revolutionary system for leearning assembly language programming on the IBM PC, The Visible Computer : 80286.

TVC:80286 is a book-and-disk combination for mastering the elusive skills of 80286 assembly language. It consists of a 540 page tutorial-style manual a program that graphically simulates the inner workings of the 80286 microprocessor chip, and dozens of demonstration programs.

### HALO '88

-The Graphics Toolkit for Contemporary Software Developers-

Already the fastest and most powerful graphics toolkit on the market, the new HALO delivers subroutines and device support for exciting, contemporary applications in publishing, office automation, vision, and image processing.

HALO'88 is a device independent library of 190 graphics subroutines. It is compatible with 18 programming languages, and over 140 hardware devices such as image scanners; graphics, vision, and imaging boards; printers and plotters; and mice. HALO'88 is designed for the complete IBM compatible microcomputer line including the PS/2 and VGA.

### MATRIX LAYOUT

The First True Computer Aided Software Engineering (CASE) Tool For The PC.

If you've ever wanted an easy, time-saving way to create your own programs without programming, it's here. It's the new Matrix Layout, for the PC and PS/2.

With this revolutionary tool, you can quickly and easily create programs that do exeactly what you want - without writing a single line of program code.

If you're a programmer, you can stop throwing away countless hours of hard, tedious work. And finally take your programs out of your imagination and put them on the computer screen.

If you're a non-programmer, you can stop throwing away ideas for programs you'd like to have "someday".

- 각 제품에 대한 상세한 자료가 준비되어 있사오니, 많은 문의 바랍니다.
- 계약 후 제품인도는 한달 안에 이루어집니다.
- 제품에 대한 품질보증은 각 VENDOR 의 계약 조건에 따릅니다.
- 본 광고에 게재되지 않은 1000 여 개의 소프트웨어에 대해서도, 전화문의를 주시거나 당사를 방문하시면, 자료를 전해 드리겠읍니다.

주식회사 **라이프보트**

서울특별시 강남구 역삼동 678-39
(한국고등교육재단 빌딩 6층)
TEL : (02) 555-1392 FAX : (02) 555-9250

퍼스널컴퓨터를 이용한
# 취업정보안내 프로그램

고대식/ 경희대학교 전자공학과 박사과정

**실제로 사회 각 분야에 걸쳐 유용한 업무용 프로그램들이 사용되고 있다. 여기에 소개하는 취업정보안내 프로그램도 그중의 하나로 이를 잘 이용하면 여러분야에서 활용이 가능하며 프로그램 공부에도 도움이 될 것이다.**

대학의 취업문은 갈수록 좁아져 가면서 한편 세분화되고 있다.

날로 다분화되고 있는 전공분야와 업계, 그리고 인구급증이라는 우리사회의 현실로 인해 취업정보는 갈수록 방대해지고 다양화되어 이에 따르는 취업정보관리·이용 또한 전산화를 통해 효과적인 운영방안이 모색되어야 한다.

필자는 대학의 취업정보운영, 관리의 전산화에 관심을 가져 취업정보를 기업체현황, 기업별 채용정보, 졸업동문 취업현황, 구직정보, 진료지도 정보 등으로 분류하여 데이터베이스화 하였고 이에 대한 관리와 이용을 효율적으로 운용하기 위한 프로그램을 개발하였다.

## 서론

1980년대부터 시작된 취업난은 공장의 자동화를 비롯한 사회적, 경제적 변화에 의하여 날이갈수록 가중되고 있고 취업을 위한 대학 졸업생들의 경쟁은 매우 치열해지고 있다. 최근 대학생들의 가장 커다란 고민중의 하나는 진로 및 취업문제이며 졸업을 앞둔 학생들은 친구나 선배 그리고 교수님들을 만나보거나 취업보도실, 상담실 등을 찾으며 취업정보를 얻으려고 애쓰고 있다. 그러나 기존 취업정보 운영체제는 관리와 이용면에서 볼 때 다음과 같은 몇가지 단점을 지니고 있었다.

① 필요한 시기에 정보를 취득하기 어렵다.

기업체 홍보물이나 기타 방법에 의하여 수집된 취업정보는 현재 그대로 열람용으로 비치하거나 상당기일 후에 재정리하여 인쇄물로 발간되기 때문에 학생들이 그 정보를 취득할 때는 이미 많은 부분이 효용가치가 없는 과거의 정보가 되기 쉬우며 학생들에게 배포되는 시기 또한 학생들의 요구에 부응하기가 어려웠다. 그러므로 학생들이 학교의 취업정보과를 이용하는 비율은 매우 적으며 취업정보 관련 전문잡지인 코리아 리쿠르트의 설문조사 자료를 보더라도 취업정보 취득경로는 잡지(22.1%), 신문(20%), 기업홍보인쇄물(17.7%), 선배(12.3%), 친구(9%), 학교 취업보도실(6%) 순으로서 학교 이용률이 제일 낮은 것으로 나타났다.

② 대량의 취업정보를 관리하기 어렵다.

국내의 그룹, 중소기업, 은행, 연구소, 공사, 외국인 회사 등을 총망라해야 할 취업정보의 양은 엄청난 것이며 그 중에서 일부분만을 정리한다하더라도 그 업체의 현황, 채용방법, 졸업생 취업현황 등 학생들이 원하는 정보의 양은 매우 방대할 것이다. 더구나 이 정보들을 새로 수정한다거나 삭제 혹은 추가하는 등의 관리는 막대한 인력과 시간을 요하게 될 것이며 학생들이 이용하기에도 매우 비효율적이다.

이와같은 현 체재의 전산처리를 고려할 때 컴퓨터에 의한 단점을 컴퓨터의 특성상 신속성, 정확성, 대량성을 만족할 수 있기 때문에 관리자의 측면에서도 효율적이며 이용자의 측면에서도 매우 편리한 기법이 될 것이다.

## 취업정보 전산화기법

### (1) 이론적 배경 (데이터베이스 및 DBASEIII+)

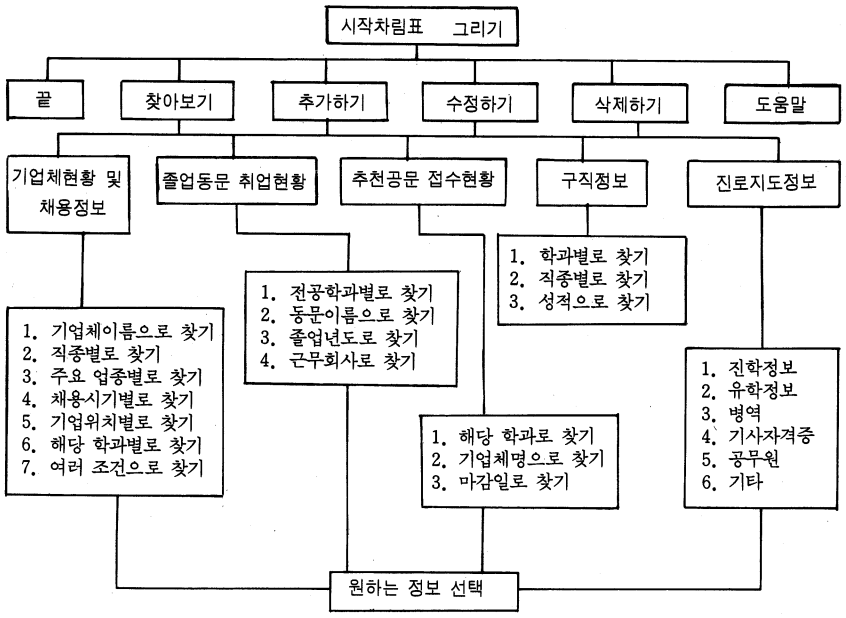
취업정보 전산처리기법은 최근 다방면에서 널리 이용되고 있는 데이터베이스론에서 그 아이디어를 찾을 수 있다. 데이터베이스(DataBase)는 데이터들의 집합체로서 데이터의 중복을 최소화하고 서로 관련있는 데이터들을 유기적으로 통합함으로서 데이터의 검색, 수정, 삭제, 추가 등을 용이하게 처리할 수 있도록 하는 시스템이며 요즈음과 같이 정보의 활용이 강력히 요구되고 있는 시점에서는 매우 중요한 것이다. 그리고 데이터베이스 관리시스템(DBMS)은 데이터베이스를 보다 합리적으로 유지, 관리하기 위한 것으로서 사용자의 요구에 따라 다양한 응용프로그램을 구비하고 있다. 이와같이 데이터베이스는 기존의 취업정보 운영상의 단점이었던 대량성의 정보를 관리하는데 있어서 정보의 중복을 피할 수 있고 모든 사용자가 정보를 취득할 수 있으며 관리자의 측면에서도 매우 효율적인 것이다.

한편 취업정보는 대부분이 한글이기 때문에 한글을 사용할 수 있는 데이터베이스용 언어가 필요한데 이를 위하여 한글코드가 맞는 DBASE Ⅲ+를 선택하면 적절히 이용될 수 있으며 필자의 경우는 고려시스템 AT를 사용하였다.

## (2) 프로그램의 흐름도

비교적 적은 비용으로 사용자가 원하는 정보를 신속·정확하게 제공해줄 수 있도록 취업관련 정보를 6개 항목으로 분류하였다. **그림 1**은 전체 흐름도이고 **화면 1**은 초보자라도 쉽게 조작할 수 있도록 마련한 시작 차림표를 보인 것이다(**리스트** 1 참조).

〔그림 1〕 전체 흐름도




## (3) 프로그램의 실제

### ① 찾아보기 프로그램

시작 차림표 중에서 제일 중요한 것은 찾아보기작업으로 각 항목별로 내용을 살펴보면 다음과 같다.

**• 기업체현황 및 채용정보**

그룹사 및 회사별 채용정보 중에서 원하는 부분만을 신속하고 일목요연하게 찾아볼 수 있도록 **화면** 2와 같이 그룹사별, 지역별, 직종별, 해당학과별, 채용 시기별로 차림표(컴퓨터 모니터에 각 항목을 나타내고 사용자의 선택여부를 묻는 형식의 메뉴)를 준비하였으며 이를 위한 상세한 데이터를 준비하여 기업을 선택하거나 취업을 준비하는데 커다란 도움이 될 수 있도록 하였다(**리스트** 2, 3 참조).

〔화면 1〕 시작 차림표

```
**************************
*    취 업 정 보 의  차 림 표    *
**************************

        [0]  작 업  끝

        [1]  찾 아 보 기

        [2]  도   움   말


        [3]  추 가 하 기
        [4]  수 정 하 기
        [5]  삭 제 하 기

 << 원 하 는  작 업 의  번 호 를  선 택 하 십 시 오.>> [1-5]:
```

**• 졸업동문 취업현황**

취업에 도움을 주는 실질적인 정보는 홍보물 등의 인쇄매체가 아니라 그 기업의 사원, 특히 졸업동문을 통하여 얻는 것이 효과적이며 채용에 있어서도 선배의 구체적인 조언, 나아가서는 선배의 직위에 따라서 학생들에게 큰 힘이 될 것으로 판단하여 졸업동문에 대한 상세한 취업현황을 정보화하였다.

**• 추천공문 채용정보**

기업체로부터 얻어지는 추천공문의 대부분은 제출서류가 많고 마감일이 촉박하기 때문에 해당학과의 학생들에게 많은 불편함이 있었다. 따라서 추천공문을 접수하는대로 우선 구직정보란에 수록된 데이터를 검색하여 해당 학생과 해당 학과에 신속히 전달할 수 있도록 하였다.

**• 구직정보 및 진로지도정보**

구직정보는 위에서 설명한 추천공문에 지도교수나 학과에서 신속히 대체할 수 있고 만약 각 기업체에 홍보하여 상호 협조를 한다면 기업체가 직접 본 시스템을 이용하여 원하는 학생을 채용할 수 있을 것이다.

이 외에도 대학원 진학정보, 공무원 시험정보, 자격증 시험정보, 유학 및 병역관계정보 등은 대학생들에게 반드시 필요한 정보이므로 이에 대한 각종 자료를 정보화 함으로써 취업의 기회를 보다 확대할 뿐만아니라 진료지도에도 일익을 담당할 수 있도록 하였다.

② 여러 조건으로 찾기

컴퓨터를 전혀 모르는 사람이라도 손쉬운 조작만으로 원하는 정보, 특히 여러가지 조건을 만족하는 정보를 얻도록 하기 위하여 찾는 방법중에서 '여러 조건으로 찾기' 프로그램을 추가하였다(리스트 4 참조).

③ 기타 프로그램

찾아보기 이외에도 도움말, 추가하기 작업, 수정하기 작업, 삭제하기 작업 등 이용과 관리에 필요한 프로그램들이다(리스트 5, 6 참조). 화면 3은 기업이름='국민은행'을 입력시켰을 때 출력된 채용정보현황이고 화면 4는 학과명='전자'를 입력시켰을 때 출력된 졸업동문 취업현황의 일부이다.(그밖의 메뉴선택이 필요한 루틴들은 리스트 7 이후부터 나타내었으므로 참고하기 바란다.)

〔화면 2〕 각 메뉴마다 자세한 선택화면을 얻을 수 있다.

```
                 ************************
                 *    찾아보기의 차림표    *
                 ************************

                 [0] 원래의 차림표로

                 [1] 기업체 현황 및 채용정보
                 [2] 졸업동문 취업현황
                 [3] 취업추천서 접수현황
                 [4] 취업희망 정보
                 [5] 진로지도 정보


         < 찾고싶은 정보의 번호를 선택하십시오.>[1-7]:
```

```
    << 기업체 현황과 채용정보 항목은 아래와 같읍니다. >>
--------------------------------------------------------------
1.기업체명  2.채용담당부 3.전화     4.설립일    5.창업주
6.매출액     7.종업원수  8.본사주소 9.공장주소 10.주요업종
--------------------------------------------------------------
1.채용시기  2.채용인원    3.채용직종 4.해당학과  5.전형방법
6.면접방법   7.급여       8.성별
--------------------------------------------------------------

 * 아래차림표에서 찾고싶은 방법을 선택하십시오.[0-7]: 0

                 [0] 원래의 차림표로
                 [1] 기업,그룹명 찾기
                 [2] 모집직종별로 찾기
                 [3] 해당학과별로 찾기
                 [4] 채용시기별로 찾기
                 [5] 주요업종별로 찾기
                 [6] 회사위치별로 찾기
                 [7] 여러조건으로 찾기
```

## 결론

이번에 소개한 취업정보 운영의 전산처리가 시행될 경우 기대될 수 있는 효과를 요약해 보면 다음과 같다.

- 취업정보의 운영자는 변화성을 수반하는 대량성의 취업정보를 효율적으로 관리할 수 있다.
- 학생들은 간단한 조작만으로도 원하는 다양한 취업정보를 신속 정확하게 제공받을 수 있다.
- 본 시스템을 통하여 학생들은 취업기회를 확대할 수 있으며 취업준비의 내실화를 통하여 취업준비생들의 취업에 크게 기여할 것이다.

〔화면 3〕 기업체명='국민은행'으로 입력시켰을 때의 정보출력 예

```
page no.            1
                              << 기업체현황 및 채용정보 >>

입력시킨 기업체명= 국민은행
==========================================================
                    <기업체명 = 국민은행>

전    화 =   771-40
채용담당부=
창 업 주 =   김욱티
설 립 일 =   1963.2.1
매 출 액 =   자 822억
종업원수 =   11450
본사주소 =   중구 서대문로2가 9-1
공장주소 =
주요분야 =   은행
해당학과 =   경상계,법학과
채용시기 =   11
채용직종 =   사무직
채용인원 =   60
채용방법 =   서류,영어,전공,면접
면접방법 =
급    여 =   380000,800%
성    별 =
기타사항 =
------------------------------------------------------
```

〔화면 4〕 졸업동문 취업현황 이용예

```
page NO.    1
                              << 졸업동문 취업현황 >>

입력시킨 학과명 전자
=========================================================
                         < 동문이름 = 고대식 >

전공학과 = 전자
이    름 = 고대식
졸업년도 = 1982
근무기업 = 경희대학원
근무부서 = 전자과
직    위 = 박사과정
성    별 = 남
연락처   = 0343-42-8858
---------------------------------------------------
                         < 동문이름 = 한백희 >

전공학과 = 전자
이    름 = 한백희
졸업년도 = 1981
근무기업 = 삼성전자
근무부서 = 칼라 T.V 설계실
직    위 = 대리
성    별 = 남
연락처   = 0331-30-2389
---------------------------------------------------
                         < 동문이름 = 고태성 >

전공학과 = 전자
이    름 = 고태성
졸업년도 = 1981
근무기업 = 삼성반도체
근무부서 = 연구소
직    위 = 연구원
성    별 = 남
연락처   = 44-333333
---------------------------------------------------
```

〔리스트 1〕 시작차림표 프로그램

```
*by KO DAE-SIK
*this is :start.prg
 set status off
 set path to c:
 set heading off
 set safety off
 clear all
 set talk off
 set bell off
 set delimiter off
 store  space(80) to clearline
 do while .t.
 clear
 @ 3,17 say "############################"
 @ 4,17 say "*  취 업 정 보 의  차 림 표   *"
 @ 5,17 say "############################"
 @ 8,22 say "[0] 작 업 끝"
 @ 10,22 say "[1] 찾 아 보 기"
 *@ 11,22 say "[2] 사용자 임의로 찾기"
 @ 12,22 say "[2] 도 움 말"
 @ 15,22 say "[3] 추 가 하 기"
 @ 16,22 say "[4] 수 정 하 기"
 @ 17,22 say "[5] 삭 제 하 기"
 store  6 to selectnum
 do while selectnum<0 .or. selectnum>5
   store " " to select
   @ 20,9 say "<< 원하는 작업의 번호를 선택하십시오.>> [1-5]:"
   @ 20,55 get select picture "#"
   read
   store val(select) to selectnum
   enddo
     do case
       case selectnum=0
         clear all
         set delimiter on
         set bell on
         use
         cancel
       case selectnum=1
         do loca
       case selectnum=2
         do help
```

```
    use
   case selectnum=3
    do add
    use
   case selectnum=4
    do edit
   case selectnum=5
```

```
     do del
    *case selectnum=6
     *do del
  endcase
  enddo t
  clear all
*eof:start.prg
```

〔리스트 2〕 채용정보 찾아보기

```
* program :loca-prg
  set heading off
  set safety off
  set talk off
  set bell off
  set delimiter off
  store space(80) to clearline
do while .t.
   clear
@ 3,22 say "**********************"
@ 4,21 say "*  찾아보기의 차림표  *"
@ 5,22 say "**********************"
  @ 8,21 say  "[0] 원래의 차림표로  "
  @ 10,21 say "[1] 기업체 현황 및 채용정보 "
  @ 11,21 say "[2] 졸업동문 취업현황 "
  @ 12,21 say "[3] 취업추천서 접수현황"
  @ 13,21 say "[4] 취업희망 정보 "
  @ 14,21 say "[5] 진로지도 정보"

 store 7 to selectnum.
*
do while selectnum <0 .or. selectnum >6
  store " " to select
  @ 18,12 say "< 찾고싶은 정보의 번호를 선택하십시오.>[1-7]:"
  @ 18,58 get select picture "#"
read
store val(select) to selectnum
enddo
```

```
*
do case
  case selectnum=0
     clear all
     set delimiter off
     set bell off
     set talk off
     use
     return
*
  case selectnum=1
*
    do chayoung
  case selectnum=2
    do donmun
  case selectnum=3
    do gongmun
  case selectnum=4
    do job
  case selectnum=5
    do jinro
*  case selectnum=6
 *   do user
endcase
enddo t
clear all
*
**end*
```

〔리스트 3〕 기업체 현황 및 채용정보

```
* program :chayoung.prg
  set heading off
  set safety off
  set talk off
  set bell off
  set delimiter off
 * store space(80) to clearline
do while .t.
    clear
```

```
@ 1,8 say "<< 기업체 현황과 채용정보 항목은 아래와 같습니다. >>"
@ 2,2 say " ------------------------------------------------------------"
@ 3,2 say "  1.기업체명 2.채용담당부 3.전화      4.설립일     5.창업주     "
@ 4,2 say "  6.매출액       7.종업원수 8.본사주소 9.공장주소 10.주요업종  "
@ 5,2 say " ------------------------------------------------------------"
@ 6,2 say "  1.채용시기 2.채용인원    3.채용직종 4.해당학과   5.전형방법  "
@ 7,2 say "  6.면접방법    7.급여       8.성별 "
@ 8,2 say " ------------------------------------------------------------"
@ 11,5 say "* 아래차림표에서 찾고싶은 방법을 선택하십시오.[0-7]: "
```

```
      @ 13,22 say "[0] 원래의 차림표로 "
      @ 14,22 say "[1] 기업,그룹명 찾기 "
      @ 15,22 say "[2] 모집직종별로 찾기 "
      @ 16,22 say "[3] 해당학과별로 찾기 "
      @ 17,22 say "[4] 채용시기별로 찾기 "
      @ 18,22 say "[5] 주요업종별로 찾기 "
      @ 19,22 say "[6] 회사위치별로 찾기 "
      @ 20,22 say "[7] 여러조건으로 찾기 "
     store 8 to selectnum
    *
    do while selectnum <0 .or. selectnum >7
      store " " to select
      @ 11,58 get select picture "#"
     read
    store val(select) to selectnum
    enddo
    *
    do case
      case selectnum=0
         clear all
         set delimiter off
         set bell off
         set talk off
         use
         return
    *
      case selectnum=1
    *
        do group1
      case selectnum=2
        do kind1
      case selectnum=3
        do part1
      case selectnum=4
        do time1
      case selectnum=5
        do field1
      case selectnum=6
        do addr1
      case selectnum=7
        do user
    endcase
    enddo t
    clear all
```

〔리스트 4〕 여러 조건으로 찾기

```
**********user
set safety off
set heading off
set delimiter off
set bell on
set margin to 1
 store .t. to ppp
 do while ppp
 store .t. to www
  do while www
use chayoung
* goto top
 store 1 to pagenum,datanum
 store "<< 기업체 현황 및 채용정보 >>" to pagehdg
 clear
@ 4,16 say pagehdg
 store " " to dd
  @ 8,20 say "[0] 원래의 차림표로 "
  @ 10,20 say "[1] 주요업종 + 해당학과 "
  @ 11,20 say "[2] 주요업종 + 채용시기 "
  @ 12,20 say "[3] 해당학과 + 채용시기 "
  @ 13,20 say "[4] 업종 + 학과 + 시기  "
@ 17,12 say "* 원하는 조건의 번호를 선택하십시오.[1-4]:" get dd
 read
clear
                if dd="0"
   use
  return
   endif
               if dd="1"
@ 2,10 say "<< 기업체현황 및 채용정보 작업중 >> "
@ 4,1 say " -------------------< 주요업종에 >-----------------------"
@ 6,3 say " 1.공사.일반단체   2.철강.기계          3. 석유.화학공업  "
@ 7,3 say " 4.건설건자재        5. 은행.증권.보험.6.섬유.의류        "
@ 8,3 say " 7.식품.음료.주류     8.제약.화장품          9.무역업          "
@ 9,3 say " 10.언론.출판.제지   11.광고.정보서비스 12.관광.운수.유통"
@ 10,3 say "  13.전기.전자.정밀 14.기타제조업        15. 외국인회사 "
@ 12,1 say "--------------------------------------------------------"
 store "                  " to mfield
 @ 15,10 say " * 찾는 업종을 두세자 입력 :" get mfield
  read
  store trim(mfield) to mfield
 store "                  " to mpart
 @ 17,10 say " * 학과명을 두세자로 입력   :" get mpart
  read
  store trim(mpart) to mpart
if len(mfield)=0 .and. len(mpart)=0
clear
exit
endif
     locate for upper(mfield) $ upper(field) .and. upper(mpart) $ ↲
      upper(part)
   if eof()
@ 20,12 say " * 잘못 입력시켰거나 찾는 정보가 없읍니다."
```

```
 read
 clear
 exit
 endif
           do print1
enddo www
endif
          if dd="2"
 store .t. to opp
 do while opp
  store "  " to mtime
  store "       " to mkind
 @ 2,12 say pagehdg
 @ 4,1 say "-----------------------< 직  종  예 >----------------"
 ? "1. 영업    2. 관리    3. 기술    4. 연구  5. 판촉    6. 생산 "
 ? "7. 사무    8. 무역    9. A/S직  10.기획   11. 개발   12. 품질관리"
 ? "13. 경리  14. 전산   15. 기타                                  "
 ? "-------------------------------------------------------------"
  @ 16,12 say "채용직종은 ?  " get mkind
   read
  store trim(mkind) to mkind
  @ 18,12 say "채용시기는 ?  " get mtime
  read
  store trim(mtime) to mtime
 if len(mkind)=0 .and. len(mtime)=0
 clear
 return
 endif
  goto top
  locate for upper(mkind) $ upper(kind) .and. upper(mtime) $ ↲
 upper(time)
  if eof()
  @ 20,20 say "잘못입력시켰거나 찾는 정보가 없읍니다."
  read
 clear
 enddo opp
 * exit
 endif
  do print1
 exit
 enddo ppp
   endif
           if dd="3"
  store .t. to lpp
 do while lpp
 store "  " to mtime
  store "      " to mpart
 @ 2,12 say pagehdg
  @ 12,12 say " 해당학과는 ? " get mpart
   read
   store trim(mpart) to mpart
  @ 14,12 say " 채용시기는 ? " get mtime
  read
  store trim(mtime) to mtime
 if len(mpart)=0 .and. len(mtime)=0
 clear
 return
 endif
 locate for upper(mpart) $ upper(part) .and. upper(mtime) $ ↲
 upper(time)
   if eof()
 @ 20,20 say "잘못입력시켰거나 찾는정보가 없읍니다. "
   read
 clear
 enddo lpp
  endif
           do print1
  exit
   enddo ppp
 endif
            if dd="4"
 goto top
  store .t. to hpp
 do while hpp
  store "  " to mtime
  store "          " to mpart
  store "          " to mfield
 @ 5,12 say pagehdg
 @ 12,12 say "주요업종은 ? " get mfield
  read
  store trim(field) to mfield
 @ 14,12 say "해당학과는 ? " get mpart
  read
  store trim(mpart) to mpart
 @ 16,12 say "채용시기는 ? " get mtime
  read
  store trim(mtime) to mtime
 if len(mtime)=0 .and. len(mpart)=0 .and. len(mfield)=0
 clear
 return
 endif
 locate for upper(mfield) $ upper(field) .and. upper(mpart) $ ↲
 upper(part) .and. upper(mtime) $ upper(time)
   if eof()
  @ 20,20 say "잘못 입력시켰거나 찾는정보가 없읍니다. "
  read
  clear
 enddo hpp
 endif
  do print1
 exit
   enddo ppp
  endif
 enddo ppp
     * enddo
      *endif
```

〔리스트 5〕 수정하기 프로그램

```
****edit-program
set heading off
set safety off
clear
set talk off
set intensity on
set bell on
set delimiter off
store .t. to ddd
do while ddd
  store " " to as
clear
@ 5,19 say "************************"
@ 6,19 say ">>  수정하기 차림표  <<"
@ 7,19 say "************************"
@ 9,20 say "[0]  원래의 차림표로 "
@ 11,20 say "[1]  기업체현황및 채용정보"
@ 12,20 say "[2]  졸업동문 취업현황"
@ 13,20 say "[3]  취업희망정보"
* @ 14,20 say "[4]  진로지도정보 "
@ 17,12 say "* 수정할 정보의 번호를 선택하십시오.[1-3]; "  get as
read
clear
if as="0"
use
return
endif
if as="1"
use chayoung
* index on group to chal
USE CHAYOUNG INDEX CHA1
store .t. to www
clear
 store " "to gg
@ 12,5 say "수정할 기업체명을 알고있읍니까 ?  [Y/N] " get gg
 read
 if gg="n".or. gg="N"
 goto top
display next 1500 group
 read
  else
endif
do while www
clear
@ 8,2 say "기업체 현황 및 채용정보를 수정중입니다."
? "=============================================="
store "                " to mgroup
  @ 15,10 say "수정할 기업체명을 입력시키시오."  get mgroup
 read
  store trim(mgroup) to mgroup
 if len(mgroup)=0
clear
exit
 endif
 find &mgroup
 clear
IF EOf()
 @ 15,10 say "잘못입력하셨거나 찾는 기업이 없읍니다."
read
 use
clear
exit
 ENDIF
@ 1,42 say "< 기업체 현황및 채용정보를> "
@ 2,42 say "< 수정중입니다.최소하려면 >"
@ 3,42 say "< Esc 키를 치십시오. >"
@ 2,5 say "기업체명:  " get group
@ 3,5 say "채용담당부:" get boo
@ 4,5 say "전    화:  " get tel
@ 5,5 say "설 립 일:  " get day
@ 6,5 say "창 업 주:  " get owner
@ 7,5 say "매 출 액:  " get don1
@ 8,5 say "종업원수:  " get number
@ 9,5 say "본사주소:  " get addr1
@ 10,5 say "공장주소:  " get addr2
@ 11,5 say "주요업종:  " get field
@ 12,5 say "채용시기:  " get time
@ 13,5 say "해당학과:  " get part
@ 14,5 say "모집분야:  " get kind
@ 15,5 say "채용인원:  " get inwon
@ 16,5 say "채용방법:  " get method
@ 17,5 say "면접방법:  " get face
@ 18,5 say "급    여:  " get don
@ 19,5 say "성    별:  " get sex
@ 20,5 say "기타사항:  " get etc
read
 clear gets
 store " " to xx
@ 21,10 say "계속하려면 'enter'키를,끝내려면 '0'자를 치십시오." get xx
  read
clear
  if xx="0"
  use
exit
 endif
enddo www
endif
      if as="2"
use donmun
index on name to don1
 use donmun index don1
 store .t. to zzz
 do while zzz
 store "            " to mname
```

```
clear
@ 8,2 say " 졸업동문현황의 수정중입니다. "
? " =================================="
 @ 15,2 say "수정할 동문이름을 입력하시오." get mname
 read
store trim(mname) to mname
   if len(mname)=0
   clear
   exit
   endif
find &mname
 clear
 if eof()
@ 15,10 say " 잘못입력하였거나 찾는 이름이 없읍니다."
read
 use
exit
endif
clear
 @ 2,42 say "< 졸업동문현황을 수정중입니다.> "
 @ 3,42 say "< 최소하려면 Esc키를 치십시오.>"
  @ 4,5 say "학    과: " get part
  @ 5,5 say "이    름: " get name
  @ 6,5 say "졸업년도: " get year
  @ 7,5 say "근 무 처: " get group
  @ 8,5 say "근무부서: " get boo
  @ 9,5 say "직    위: " get grade
  @ 10,5 say "성    별: " get sex
  @ 11,5 say "전    화: " get tel
  read
clear gets
 store " " to xx
@ 15,8 say "계속하려면 'enter'키를,끝내려면 '0'자를 치십시오." get xx
read
clear
if xx="0"
 use
exit
  endif
clear
enddo zzz
endif
if as="3"
use job
index on name to job1
use job index job1
 store .t. to ggg
 do while ggg
 clear
store "              " to mnam
 clear
@ 8,2 say " 취업희망 카드를 수정중입니다."
@ 15,2 say "관련카드중 수정할 이름을 입력하십시오." get mnam
read
store trim(mnam) to mnam
if len(mnam)=0
clear
exit
continue
endif
find &mnam
clear
if eof()
@ 15,5 say "잘못 입력하셨거나 찾는 정보가 없읍니다."
clear
read
use
exit
endif
clear
@ 2,42 say "< 취업희망카드를 수정중입니다.> "
@ 3,42 say "< 취소하려면 Esc키를 치십시오.>"
@ 4,5 say "성    명:" get name
@ 5,5 say "희망업종:" get field
@ 6,5 say "소    속:" get part
@ 7,5 say "가족관계:" get goon
@ 8,5 say "현 주 소:" get addr
@ 9,5 say "전    화:" get tel

@ 11,5 say "경    력:" get carr
@ 12,5 say "자 격 증:" get card
@ 13,5 say "희망직위:" get addr1
@ 14,5 say "특    기:" get main
@ 15,5 say "종    교:" get faith
@ 16,5 say "기타사항:" get etc
read
clear gets
store " " to pp
@ 19,3 say "계속하려면 'enter'키를 끝내려면 '0'자를 치십시오." get ♪
pp
read
clear
if pp='0'
use
exit
endif
clear
enddo ggg
endif
      if as='4'
do edit1
endif
enddo
clear
enddo ppp
*********************************
```

〔리스트 6〕 삭제하기 프로그램

```
************ del program
set heading off
set safety off
clear
set talk off
set intensity on
set bell on
set delimiter off
store 1 to datanum,pagenum
store .t. to ppp
do while ppp
clear
  store " " to as
@ 3,23 say "************************"
@ 4,23 say "> 정보삭제하기 차림표 <"
@ 5,23 say "************************"
@ 8,25 say "[0] 원래의 차림표로 "
@ 10,25 say "[1] 기업체 현황 및 채용정보"
@ 11,25 say "[2] 졸업동문 취업현황"
@ 12,25 say "[3] 추천서 접수현황"
@ 13,25 say "[4] 취업희망카드"
@ 16,18 say "* 삭제할 정보의 번호를 선택하십시오[1-4]:" get as
read
clear
if as="0"
return
endif
     if as="1"
use chayoung
store .t. to www
do while www
clear
store " " to yy
@ 8,3 say "* 삭제할 기업체명을 아십니까 ? [Y/N]" get yy
read
if yy="N" .or. yy="n"
display next 1500 group
?
read
clear
else
endif
@ 12,3 say "<< 기업체 현황과 채용정보를 삭제하고 있습니다. >>"
 store "                    " to MGROUP
@ 15,3 say "** 삭제할 기업체이름을 입력시키십시오." get MGROUP
  read
 store trim(MGROUP) to MGROUP
 if len(MGROUP)=0
clear
exit
endif
 locate for upper(mgroup) $ group
clear
if eof()
@ 15,5 say "<< 잘못입력시켰거나 찾는기업체가 없습니다.>>"
read
use
clear
exit
endif
@ 0,1 say "<< 기업체현황과 채용정보를 삭제하고 있습니다. >> "
? "-----------------------------------------------------"
?
?
? "기업체명:   ",group
? "채용담당:   ",boo
? "전   화:   ",tel
? "설 립 일:   ",day
? "창 업 주:   ",owner
? "매 출 액:   ",don1
? "종업원수:   ",number
? "본사주소:   ",addr1
? "공장주소:   ",addr2
? "주요직종:   ",field
? "해당학과:   ",part
? "채용시기:   ",time
? "채용직종:   ",kind
? "채용인원:   ",inwon
? "채용방법:   ",method
? "면접방법:   ",face
? "급   여:   ",don
? "성   별:   ",sex
? "기타사항:   ",etc
 store " " to xx
@ 21,3 say "<< 삭제하려면 'enter'키를 차림표로 가려면 '0'자를 치십시오.>> " get xx
 read
clear
if xx="0"
exit
endif
 delete
 pack
 store " " to gh
  @ 15,5 say "<< 계속하려면 'enter'키를 끝내려면 '0'자를 치십시오.>>" get gh
  read
clear
 if gh="0"
use
 exit
  endif
enddo www
endif
          if as="2"
```

```
use donmun
clear
store .t. to zzz
do while zzz
  clear
*display next 1000 name,part
@ 2,12 say "<< 졸업동문현황을 삭제중입니다. >> "
store "                    " to bo
@ 12,5 say "삭제할 동문이름을 입력시키십시오." GET BO
READ
store trim(bo) to bo
if len(bo)=0
clear
exit
endif
 locate for upper(bo) $ upper(name)
clear
IF EOF()
 @ 15,5 say "잘못입력하였거나 찾는이름이 없습니다."
read
clear
use
exit
ENDIF
? "<< 졸업동문현황을 삭제하고 있읍니다. >> "
?
?
 ? "이    름:  ",name
 ? "전공학과:  ",part
 ? "졸업년도:  ",year
 ? "기업체명:  ",group
 ? "근무부서:  ",boo
 ? "직    위:  ",grade
 ? "성    별:  ",sex
 ? "전    화:  ",tel
 store " " to xx
@ 13,5 say "삭제하려면 'enter'키를 차림표로는 '0'자를 치시오." get xx
read
clear
if xx="0"
use
exit
  endif
 delete
 pack
store " " to gh
@ 15,5 say "계속하려면 'enter'키를 끝내려면 '0'자를 치십시오." GET GH
READ
clear
IF GH="0"
use
exit
endif
```

```
enddo zzz
endif
   if as="3"
use gongmun
store .t. to sss
do while sss
 display next 1000 group,time
 read
clear
@ 2,10 say " << 추천의뢰서를 삭제중입니다. >> "
store "        " to mtime
@ 12,5 say " 삭제할 마감일을 입력시키십시오. " get mtime
read
store trim(mtime) to mtime
if len(mtime)=0
clear
exit
endif
locate for upper(mtime) $ upper(time).or. upper(mtime)>upper(time)
if eof()
@ 15,5 say "잘못입력시켰거나 찾는 정보가 없읍니다."
read
clear
exit
endif
clear
do while .not. eof()
? "      << 추천의뢰서를 삭제중입니다. >> "
? "     ---------------------------------"
?
? "<< 기업체명:",group+">>, << 마감일 :",time," >>"
 ? "전공학과:" ,part
 ? "추천인원:" ,inwon
 ? "기업체명:" ,group
 ? "직    종:" ,kind
 ? "급    여:" ,don
 ? "근 무 처:" ,addr
 ? "전    화:" ,tel
 ? "제출서류:" ,paper
 ? "마 감 일:" ,time
 ? "기    타:" ,etc
 read
store " " to ww
@ 19,5 say "삭제하려면 enter 키를 차림표로 가려면 [0]자를 치십시오 ." get ww
read
if ww='0'
clear
exit
endif
delete
pack
clear
continue
```

```
enddo
store " " to fk
@ 17,5 say "계속하려면 enter 키를 끝내려면 [0] 자를 치십시오." get ♪
fk
  read
clear
  if fk="0"
use
exit
```

```
 endif
enddo sss
endif
             if as="4"
do dela
endif
enddo ppp
**********************
```

(리스트 7) '찾아보기' 프로그램

```
************group1.prg is
  set heading off
  set safety off
  clear
  set talk off
  set delimiter off
  set bell on
  set margin to 1
set exact on
clear
use chayoung
store " " to yen
@ 12,10 say "기업이름을 아십니까 ? [YES/NO] " get yen picture "!"
read
if yen="n" .or. yen="N" .OR. yen="NO" .or. yen="no"
clear
do group1la
endif
if yen="y" .or. yen="Y" .OR. yen="YES" .OR. yen="yes"
* use chayoung index dd1
 goto top
endif
store 1 to pagenum,datanum
store " << 기업체현황 및 채용정보 >>" to pagehdg
store .t. to hhh
do while hhh
store "                " to mddno
@ 15,10 say "찾고싶은 기업이름을 한글로 입력시키십시오."
@ 17,10 say "그룹을 원하면 'ㅇㅇ그룹'으로 치십시오.    " get mddno
read
  store trim(mddno) to mddno
if len(mddno)=0
clear
return
endif
 store " " to prtest
  clear
@ 3,1 say "입력시킨 기업이름 ="
@ 3,21 say mddno
*find &mddno
@ 5,1 say "작업중이오니 기다리십시오."
locate for upper(mddno) $ upper(group)
if eof()
  ? chr(27)+"찾는 기업이 없읍니다."
else
exit
endif
enddo hhh
  store " " to select
@ 7,1 say "정보를 화면으로 보고싶으면 '0'자를  "
@ 9,1 say "프린터로 출력시키고 싶으면 '1' 자를 치십시오" get select
  read
do case
  case select="0"
  clear
  store 22 to pagelen
do while .not. eof()
* ? "page no." , pagenum
   ? space(30) + pagehdg
 ? "입력시킨 기업이름 = " ,group
   ? "========================================"
   ?? "============================="
* do while .not. eof()
 ? "기업체명= "+trim(substr(group,1,20))
 ? "채용담당부"+trim(substr(boo,1,20))
 ? "전화번호= "+ tel+" [ 처누 근앵잘릉 ="+group+" ]"
 ? "설립일  = "+trim(substr(day,1,12))
 ? "창업주  = "+trim(substr(owner,1,16))
 ? "매출액  = "+trim(substr(don1,1,20))
 ? "종업원수= "+trim(substr(number,1,20))
 ? "본사주소= "+trim(substr(addr1,1,40))
 ? "공장주소= "+trim(substr(addr2,1,40))
 ? "주요직종= "+trim(substr(field,1,30))
 ? "해당학과= "+trim(substr(part,1,20))
 ? "채용시기= "+trim(substr(time,1,8))
 ? "채용직종= "+trim(substr(kind,1,20))
 ? "채용인원= "+trim(substr(inwon,1,15))
 ? "채용방법= "+trim(substr(method,1,26))
 ? "면접방법= "+trim(substr(face,1,26))
 ? "급    여= "+trim(substr(don,1,20))
```

```
  ? "성   별= "+trim(substr(sex,1,10))
  ? "기타사항= "+trim(substr(etc,1,30))
? "-----------------------------------------------------------"
 datanum=datanum+1
store " " to opp
? " 끝내려면 [0] 자를 계속하려면 [enter ]키를 치시오.:"
@ 24,54 get opp
read
if opp="0"
clear
return
endif
continue
enddo
*
case select="1"
clear
store 51 to pagelen
set console off
set print on
do while .not. eof()
  if datanum=1
  ? "page no. ",pagenum
  ? space(30)+pagehdg
 ?
 ? "입력시킨 기업체명=" ,group
 ? "=========================================="
 ?? "======================="
endif
? space(20)+"<기업체명 = "+trim(substr(group,1,20))+">"
?
? "채용담당부= "+trim(substr(boo,1,25))+"          전화 = "+tel
? "창 업 주 =  "+owner+"                설립일  =  "+day
? "매 출 액 =  "+don1 +"            종업원수=  "+number
? "본사주소 =  "+addr1
? "공장주소 =  "+addr2
? "주요직종 =  "+field
? "해당학과 =  "+trim(substr(part,1,30))
? "채용시기 =  "+trim(substr(time,1,10))
? "채용직종 =  "+trim(substr(kind,1,30))
? "채용인원 =  "+trim(substr(inwon,1,10))
? "채용방법 =  "+trim(substr(method,1,30))
? "면접방법 =  "+trim(substr(face,1,30))
? "급   여 =  "+trim(substr(don,1,20))
? "성   별 =  "+trim(substr(sex,1,10))
? "기타사항 =  "+trim(substr(etc,1,30))
? "-------------------------------------------------"
datanum=datanum+1
if datanum>3
eject
pagenum=pagenum+1
datanum=1
endif
continue
enddo
eject
endcase
release all
set print off
set console on
?
?
?
?
?
?
? "            아무키나 치십시오."
read
read
clear
return
*
* eof******** group1.prg ****************************
```

# □ 과월호 구입안내

- ■ 기  간 : 1988년12월호~1989년 5월호까지
- ■ 가  격 : 3,000원(권당)
- ■ 신청방법 : 책속에 있는 은행지로 용지를 이용하여 필요하신 호수를 기입, 본사로 신청하시면 됩니다. 송료는 본사가 부담합니다.
- ■ 문 의 처 : 월간 퍼스널컴퓨터 영업부 전화 (02)754-1044(대표)

# 타가보드

이해성/ 어플라이드 엔지니어링

**타가보드는 미국 AT&T사에서 1986년도에 개발한 그래픽보드로서 비디오 입력기능과 다양한 칼라를 낼 수 있다는 것이 특징이다. 특히 비디오 입력에는 비디오 카메라 및 VTR 등으로부터 콤포지트 및 RGB 아날로그 신호도 입력받을 수 있는데 종류에 따라서는 4096칼라에서 1,600만가지 칼라를 표현할 수 있어 에니메이션에 많이 사용되고 있다.**

## 그래픽보드

### ●그래픽보드란 무엇인가?

70년대 이후 라스터 스캔형 디스플레이 장치가 컴퓨터에 채택되면서 대중화되기 시작하여 보급된 장치로서, 모니터상에 표시할 화상자료를 메모리에 보관하였다가 적당한 방법을 이용하여 화상 데이터 신호를 모니터에 전달해주는 장치이다.

일반적으로 그래픽보드는 컴퓨터의 CPU가 그래픽보드의 메모리에 데이터를 기록하면, 그 즉시 기록된 비트 패턴에 따라 모니터에 화상이 출력되도록 되어 있다.

앞에서 언급한 바와 같이 대중적인 추세로 보급되는 라스터 스캔형은 전자빔을 화면의 위에서 시작하여 왼쪽에서 오른쪽으로 스캐닝하기 때문에 붙여진 이름이며, 디스플레이 장치는 칼라를 얻기가 용이하여 움직이는 화상을 만들기 쉽다는 장점을 지니고 있어 많이 쓰이고 있다.

이런 라스터 스캔형 모니터에는 반드시 디스플레이 메모리가 존재해야 원하는 화상을 얻을 수 있다. 따라서 컴퓨터로 그래픽 처리를 하기 위해서는 그래픽보드가 필요로 하게 된다(그림 1 참조).

반면에 벡터 스캔형은 라스터 스캔형과는 달리 화면을 라인 단위로 스캐닝하지 않고 변화를 일으키는 점 단위로 스캐닝하기 때문에 속도가 더 빠르며 적은 메모리를 사용하기 때문에 비디오 게임 등에 많이 쓰이고 있다.

### ●그래픽보드의 역할

불과 얼마안된 일이지만 과거의 컴퓨터는 연산의 결과를 단순히 프린터를 사용하거나, 모니터상에 디스플레이하는 것으로 그 결과를 표시하여 일부 전산 관련 종사자만이 사용하였으나, 70년대 이후의 컴퓨터 메이커들은 일반 유저에게 쉽게 접근할 수 있는 영상표현 방향으로 제품을 개발하여 내놓고 있다.

그 결과로 오늘날 보급된 컴퓨터에서는 처리결과를 쉽게 그림으로 표현해주는 방식을 채택하여 우리들의 일상 업무에서나 또는 설계도면의 제작, 그래픽 아트, 에니메이션 등으로 한층 친근한 벗이 되었으며, 이런 것들을 가능케한 그래픽보드의 역할은 필수적인 존재로 취급하기에 이르렀다.

## 그래픽보드의 종류

1982년 IBM사가 IBM PC를 발표하면서 MDA(Monochrome Display Adapter)라 불리우는 최초의 카드가 나왔으며, 잇달아 CGA(Color Graphic Adapter) 나왔다.

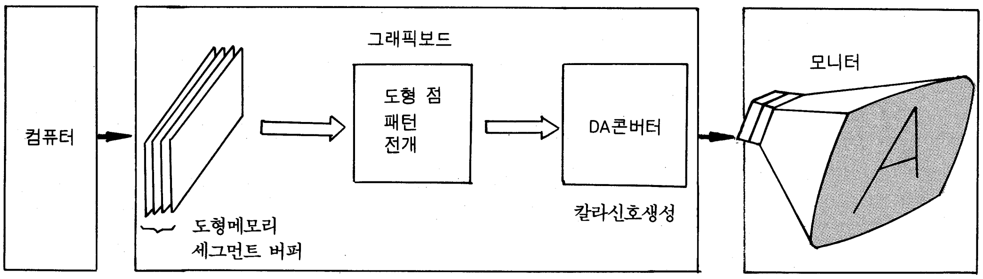
그 후에 HCT(Hercules Computer Technologies)사와 Plantronics사는 흑백과 칼라에서 높은 해상도와 다양한 색상으로 IBM PC의 표준 그래픽보드를 향상시켰다.

그 결과로 오늘날 가장 널리 쓰이고 있는 그래픽보드로는 HGC, CGA, EGA, PGA, VGA 등이 있는데, 여기서는 이들의 두드러진 특징과 해상도 등을 간략하게 알아보고 타가보드를 설명하겠다.

### ●HGC

국내의 16비트 PC가 거의 허클리스 그래픽카드를 채택하고 있어 표준카드(?)인지의 의문을 제기할 정도로 일반 유저와 친숙하게 쓰이고 있다. 본래 허클리스 그래픽카드는 IBM PC의 최초의 비디오 보드인 MDA(-Monochrome Display Adapter)가 그래픽을 지원하지 못하고 텍스트 전용모드(모니터의 화면을 문자단위로 사용하는 모드)만을 지원하는 한계로 인해 허

〔그림 1〕 그래픽이 처리되는 과정



클리스 그래픽카드가 모노크롬 모니터에 많이 쓰이고 있다.

해상도는 720×348을 제공하며 텍스트 모드에서는 한 문자당 9×14로 구성되어 있다.

● CGA

IBM PC의 CGA는 기본적으로 고해상도, 저해상도, 텍스트 모드가 있다. 이는 MDA 다음에 발표된 최초의 칼라 그래픽보드이다. 4가지 칼라를 표현하는 320×200과 흑백으로 640×200의 해상도를 갖고 있는 것이 있으며, 8×8의 텍스트를 갖고 있다.

● EGA

EGA(Enhanced Graphics Adapter) 보드는 CGA를 개선한 것으로 CGA와 호환성을 갖고 있다. 4개의 텍스트 모드를 가지고 있으며 MDA, CGA, HGC의 특징을 결합하여 64 칼라중에 16 칼라를 동시에 사용하고 640×350의 해상도를 갖고 있다.

● PGA

PGA(Professional Graphics Adapter)는 EGA 다음에 발표되어 이전에 나온 어느 보드보다 훨씬 강력한 기능을 갖고 있다. 이 보드는 256 칼라를 사용하기 때문에 보다 많은 메모리를 사용한다. 또한, PGA는 EGA보다 비싸며 고품질의 그래픽용으로 쓰인다. 해상도는 640×480을 갖고 있다.

● VGA

PS/2 시리즈가 나오면서 이 보드가 발표되었고, 뛰어난 그래픽과 칼라로 인해 유저로부터 관심을 끌고 있으며 얼마 안있어 VGA보드가 그래픽보드의 표준이 되리라 예상된다.

VGA 보드는 기존의 보드와는 달리 아날로그방식이기 때문에 모니터를 싼 가격으로 만들 수 있고 17가지 텍스트 모드가 있으며, 앞으로는 더욱 많이 사용될 것이다.

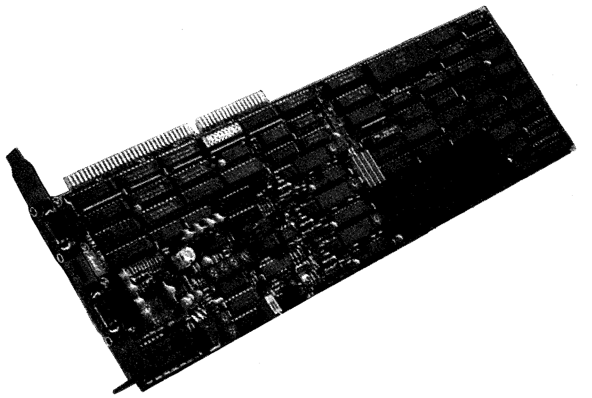
## 타가보드의 설치방법

타가보드 설치에 앞서 이 보드를 인스톨하기 전에 갖춰야 할 장비는

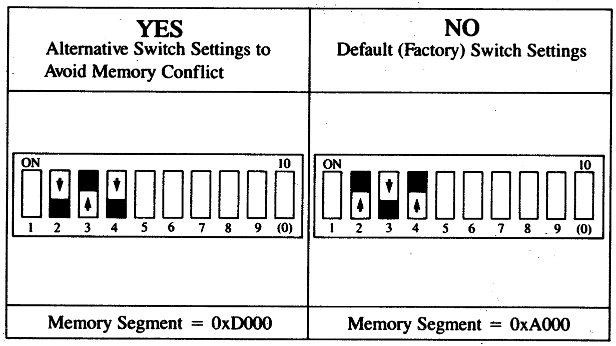
- TARGA 16
- AT&T PC 6300, 6300+, IBM PC AT/XT 또는 100% 호환되는 마이크로컴퓨터
- 디스플레이 장치
  - 아날로그 RGB 모니터와 쉴드(Shielded) RGB 케이블
  - 콤포지트 비디오 모니터와 쉴드 콤포지트 비디오 모니터
- 비디오 소스(아날로그 RGB, 콤포지트 비디오)
- 스크류 드라이버
- TARGA 데이터 디스켓
- DOS 디스켓(2.0이상)-데모 프로그램을 위해
- 256K 바이트 이상의 램
- HDD

등이 있다.

〔사진 1〕 타가 16의 외관



〔그림 2〕 다른 그래픽 어댑터를 사용할 때 메모리 세그먼트를 조정하는 예




타가보드를 설치(AT인 경우에는 16비트 슬롯에 연결)하고, EGA 보드 또는 A000:0000번지를 사용하는 보드와 병행하여 사용할 경우에는 딥 스위치를 그림 2와 같이 조정하여 메모리 세그먼트를 바꾸어야 한다.

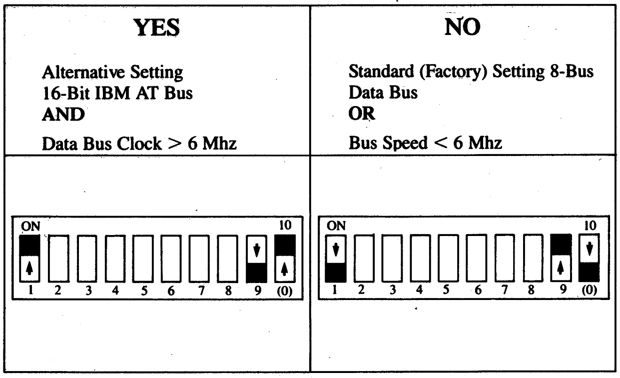
EMS(Expanded Memory Specification) 보드를 사용할 때는 메모리 세그먼트가 겹치지 않도록 딥 스위치를 조정해야 한다. EMS 보드는 C000:000, CC00:000, D000:000 등을 사용하므로 타가보드는 A000:000에 세그먼트 번지를 고정한다.

또한, 데이터 버스 클럭이 6MHz를 기점으로 크거나 작은지를 판단하여 딥 스위치를 세팅하여야 하는데 그 이유는 모니터가 데이터 신호를 받아들일 수 없기 때문이다(그림 3). 그리고 타가보드의 데이터 버스 클럭이 8MHz 이상일 경우에는 작동되지 않을 가능성이 있으므로 이를 유념할 필요가 있다.

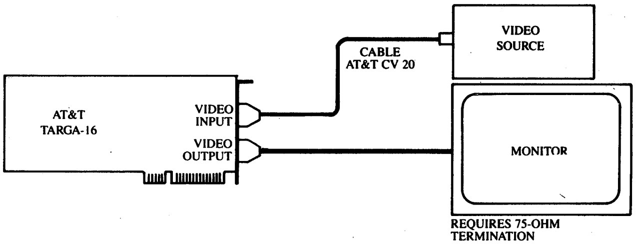
● 입출력장치의 설치

입출력장치의 설치에는 그림 4에서와 같이 비디오 소스와 콤포지트 모니터를 타가보드의 입출력단자에 각각 연결하는 경우와 그림 5처럼 비디오 소스와 아날로그 RGB 모니터를 타가보드의 입출력단자에 연결하는 방법을 들 수 있다.

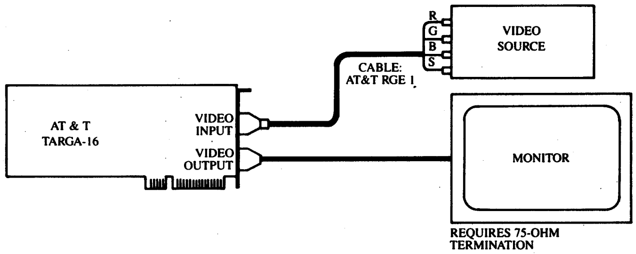
〔그림 3〕 데이터 버스 클럭에 의한 딥 스위치 조정



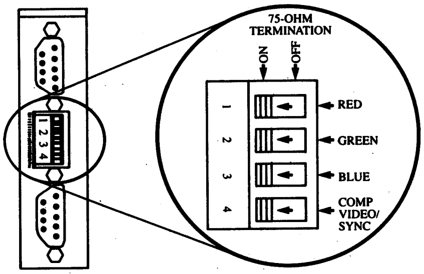

〔그림 4〕 콤포지트 비디오 입력의 연결




〔그림 5〕 아날로그 RGB 입력의 연결




〔그림 6〕 입력 임피던스의 세팅




여기서 주의할 것은 모니터와 비디오 소스를 타가 16에 연결하는데 있어 쉴드 케이블(외부로부터 영향을 많이 받을 때 이를 보호하기 위해 사용)을 항상 사용하여야 한다.

타가보드의 입출력단자 사이에 입력 터미네이션은 전기적으로 연결된 콤포지트 동기입력과 비디오 입력을 위해 하나의 스위치가 터미네이션 제어를 할지라도 RGB 입력을 위해 75Ω을 선택하기 위한 터미네이션을 3개의 스위치가 제공하고 있다(그림 6 참조).

## 타가보드의 특징

TARGA(Truevision Advanced Raster Graphics Adapter)는 미국 AT&T사에서 1986년도에 개발한 그래픽보드로서 비디오 입력기능과 다양한 칼라를 낼 수 있다는 큰 특징이 있다.

특히 비디오 입력에는 비디오 카메라와 VTR 등으로부터 콤포지트 및 RGB 아날로그 신호를 입력받을 수 있다. 또한 비디오 출력은 Genlock* 기능이 내장되어 있어 U-matic VTR 등에서 활용할 경우 본격적인 영상의 편집, 수정, 합성으로 활용할 수 있으며 실제로 KBS-TV 등에서 에니메이션에 사용되고 있다.

한가지 더 첨가한다면 타가보드는 칼라를 표현해 내는데 막강한 기능이 있어 4096 칼라에서 1,600만가지 칼라를 표현할 수 있다.

타가보드의 종류에는 표현하는 색의 수에 따라 타가 8, 타가 16, 타가 24, 타가 32, 타가 M8 등이 있으며, 칼라의 다양한 색을 만끽하기 위해서는 타가16 이상을 사용해야 한다.

### ● 타가보드의 구조

타가보드는 일반적인 그래픽 보드와는 달리 영상신호를 처리하는 특징이 있기 때문에 독특한 구조를 가지고 있다(그림 7참조).

그림 7에서 보는 바와 같이 보드의 구조중에서 주요부분은 콤포지트 비디오 디코더, A/D 콘버터, 콘트롤 레지스터, 비디오 타이밍 콘트롤러, 디스플레이 버스, 디스플레이 데이터 멀티플렉서, 디스플레이 메모리, 시프트 레지스터, D/A 콘버터, 비디오 믹서, 콤포지트 비디오 발생기 등으로 구성된다.

**(1) 콤포지트 비디오 디코더**

콤포지트 비디오 디코더는 타가 16에만 있으며 표준 NTSC 신호를 RGB 신호로 분리시키는 장치로 입력신호내의 동기신호는 출력을 Genlock시키는데 사용된다.

**(2) A/D 콘버터**

비디오 디코더로부터 얻어진 신호 또는 RGB신호를 디지탈신호로 바꾸어 주는 역할을 한다.

**(3) 디스플레이 데이터 멀티플렉서**

다음의 3가지 소스에서 디스플레이 버스로 픽셀 데이터를 선택적으로 전송하는 장치이다.

① The Diplay Memory
② The Border Color Register
③ The Analog-to-Digital Converters

**(4) 디스플레이 메모리**

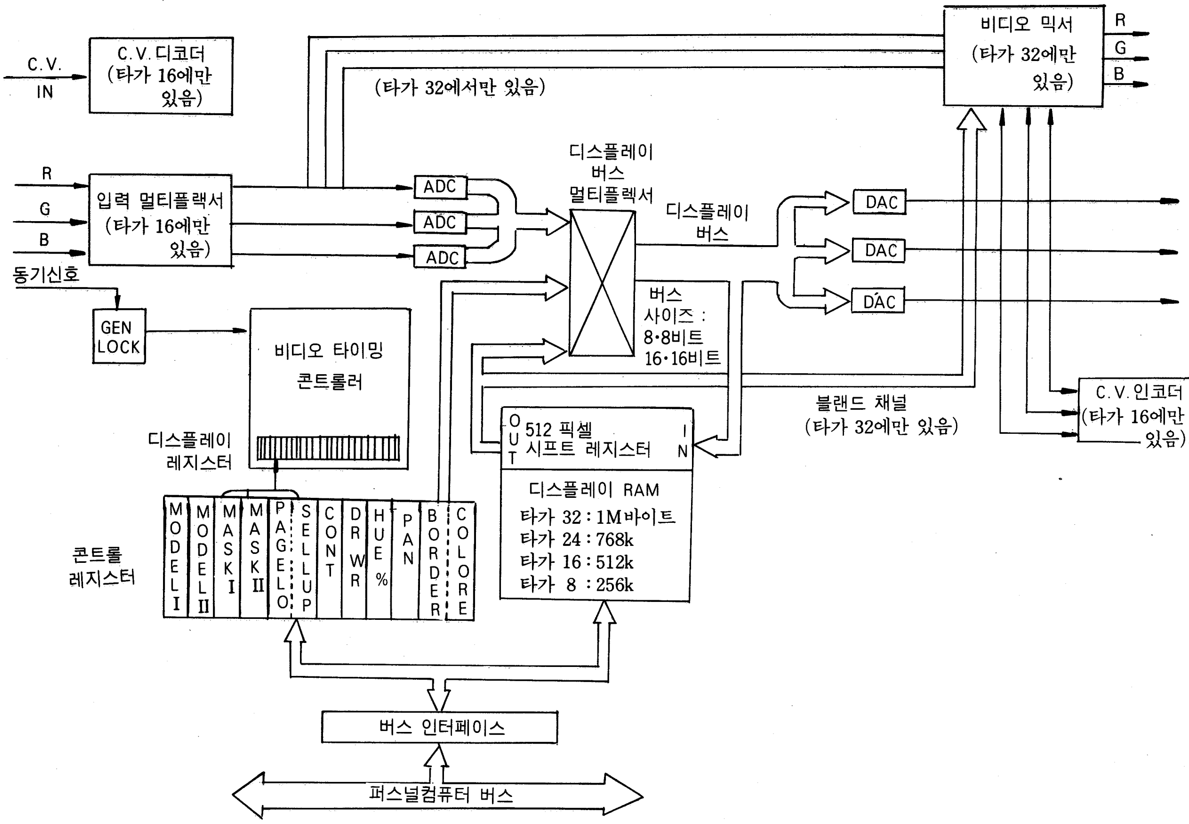
각 타가보드는 좌우로 512개의 메모리 배열을 가지고 있으며 보드 형태에 따라 사용되는 메모리에 차이가 있다.

타가 16→512K 바이트
타가 24→0.75M 바이트
타가 32→1M 바이트

**(5) D/A 콘버터**

비디오 디스플레이로 신호를 만들어주기 위해 디스플레이 버스상의 픽셀 데이터를 Red, Green, Blue 3개의 신호를 디지탈에서 아날로그신호로 바꾼다. 타가 24, 32는 8비트로 타가 16은 5비트로 D/A 변환한다.

〔그림 7〕 타가보드의 블록다이어그램




**(6) 콤포지트 비디오 발생기**

콤포지트 비디오 발생기는 타가 16에 있으며, 이 장치는 RGB와 동기신호를 표준 NTSC 콤포지트 비디오 신호로 바꾼다.

**●타가보드의 특징**

타가보드는 라이브 모드, 메모리 모드, 오버레이 모드, 블랜드 모드라는 서로 다른 모드를 가지고 있다.

타가보드의 진가는 바로 이 모드에 있다고 해도 과언이 아닐성 싶다.

**(1) 라이브 모드**

카메라나 비디오로 입력된 영상정보가 A/D 콘버터를 통하여 디스플레이 버스에 위치하면 ~~석 모니터~~에 디스플레이되는 상태를 말한다.

**(2) 메모리 모드**

디스플레이 보드내의 스캔라인에 대해 픽셀 데이터가 디스플레이 메모리에서 디스플레이 메모리 시프트 레지스터로 로드되며 보드의 칼라는 콘트롤 레지스터에 의해 지정된다. 이 모드에서는 영상신호 Capture가 안된다.

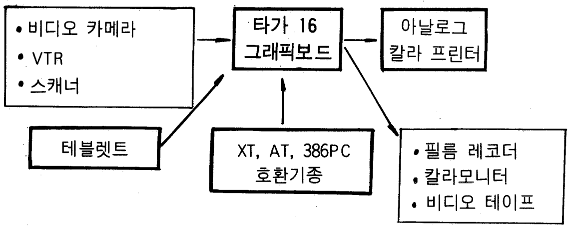
**(3) 오버레이 모드**

타가 16과 타가 32에 이 기능이 있으며, 디스플레이 버스상에 나타난 영상신호가 A/D 변환된 상태인지, 또는 디스플레이 메모리로부터 나온 상태인지를 구분할 수 있게 해준다.

오버레이 비트가 제로이면 디스플레이 메모리로부터 지정된 칼라가 디스플레이 버스상에 위치하고, 오버레이 비트가 1일 경우에는 A/D 변환 출력만 나타낸다.

쉽게 예를 들자면 TV 화면에서 자막은 움직이지 않는 상태이고, 배경의 화상은 움직이는 효과를 얻을 수 있다.

〔그림 8〕 타가보드와 연결되는 주변장치




**(4) 블랜드 모드**

타가 32에만 있는 특수기능으로 D/A 변환 아날로그 출력과 라이브 비디오 신호를 혼합하여 만든 모드이다.

즉 컴퓨터 내부에 그래픽처리된 내용과 외부 카메라 등으로 입력된 신호가 서로 혼합되게 하는 모드로서 0~127 사이의 값을 가지고 혼합정도를 선택할 수 있도록 제공한다. 중간값(64)에 대해서는 이들 두 비디오 소스를 혼합하여 동적인 영상에서도 Anti-Alias* 텍스트를 삽입할 수 있게 해준다.

**●시스템 구성방법의 예**

**(1) 아티스트를 위하여**

그림 8에서 예시한 대로 시스템을 구성하여야 사용할 수 있으나, 입출력장비가 고가이므로 개인적으로 사용하려면 굵은선

부분만 구성하고 나머지는 입출력장비를 전담으로하는 업체를 이용하는 방법도 좋겠다.

특별히 창작활동을 하는 작가들은 타가보드의 프로그램이 종료되었어도 디스플레이 메모리 내용이 지워지지 않으므로 여러가지 응용프로그램을 병행하여 활용할 수도 있다.

**(2) CAD 사용자를 위하여**

CAD 소프트웨어에서는 색처리가 미흡하므로 CAD에서 작업한 모델 또는 드로잉 화일을 타가보드가 설치된 시스템을 이용하여 다양한 칼라처리를 마음껏 할 수 있도록 되어 있다.

## 사용되는 소프트웨어와 프로그램 개발에 대해

●타가보드상에서 동작하는 소프트웨어

**(1) TIPS** :2차원 페인팅 소프트웨어(라스터 그래픽용)

**(2) RIO** :2차원 페인팅 소프트웨어(오브젝트방식)

**(3) TOPAS**:3차원 에니메이션 소프트웨어(방송용/CF제작 등에 사용)

**(4) Picture Power** :화상자료를 문서자료와 함께 데이터베이스화할 수 있는 소프트웨어

**(5) Versa CAD 및 Auto CAD 등**

●타가 프로그램 개발에 대해

타가보드를 이용하여 그래픽 프로그램을 작성하려면 Media Cybernetics사의 Halo 88이나 AT&T사의 Epic Division에서 판매하는 타가 툴을 이용하여야 하며 특히 타가 툴은 "C"언어로 작성된 라이브러리를 소스 코드로 제공하고 있다.

## 미래의 전망

타가보드는 비디오 영상을 받을 수 있는 독특한 장점 때문에 다양한 분야의 응용이 기대된다.

필자의 소견으로는 프리젠테이션용, 방송용 자막이나 영상편집, 화상데이터 베이스, 화상전송 등의 소프트웨어 등이 시급히 개발되었으면 하고 만화영화제작, 성형수술, 메이크업, CF, 슬라이드 수정 및 합성, 인물몽타즈 등등 다각적인 분야에서 활용이 기대된다.

이에 반해 국내에서 생산되는 그래픽보드들은 한글카드 몇몇 종류에 국한되고 있다. 이글을 쓰면서 국내에서 자체생산되는 막강한 그래픽보드가 없어 이를 독자에게 소개하지 못하는 점을 안타깝게 여기며, 추후로 이번 기사에 관심을 갖는 많은 업체들이 참여해 줄 것을 부탁드린다. PC

〈참고문헌〉


1. TARGA User's GUIDE / Truevision Inc
2. TARGA Technical GUIDE / Truevision Inc
3. VISTA User's GUIDE / Truevision Inc
4. 컴퓨터 그래픽스 / 고찬 저 / 서울산업대학 출판부


〈용어해설〉

*Genlock
Generator Locking의 약자로서 여러개의 비디오 소스가 있을 때 각 비디오 소스의 동기신호를 마스터 동기신호로 일치시키는 것이다. 예를 들면 화상편집기에서 마스터 동기신호와 타가보드에 입력되어 타가의 출력이 마스터 동기신호와 일치하는 것을 말한다.

*Auti-alias
컴퓨터에서 화상을 표시할 경우 사선 등에서 벽돌을 쌓놓은 듯한 눈에 거슬리는 부분을 시각적으로 일직선이 되도록하는 화면표시 방법이다.

*Hue
적색에서 시작하여 보라색으로 이어지는 일련의 무지개색 순환써클

*Saturation
채도를 말하며 백색과의 혼합정도를 말함.

*Contrast
명도를 말하며 색의 밝기와 어둡기를 말함

# □ 디버깅 룸 □

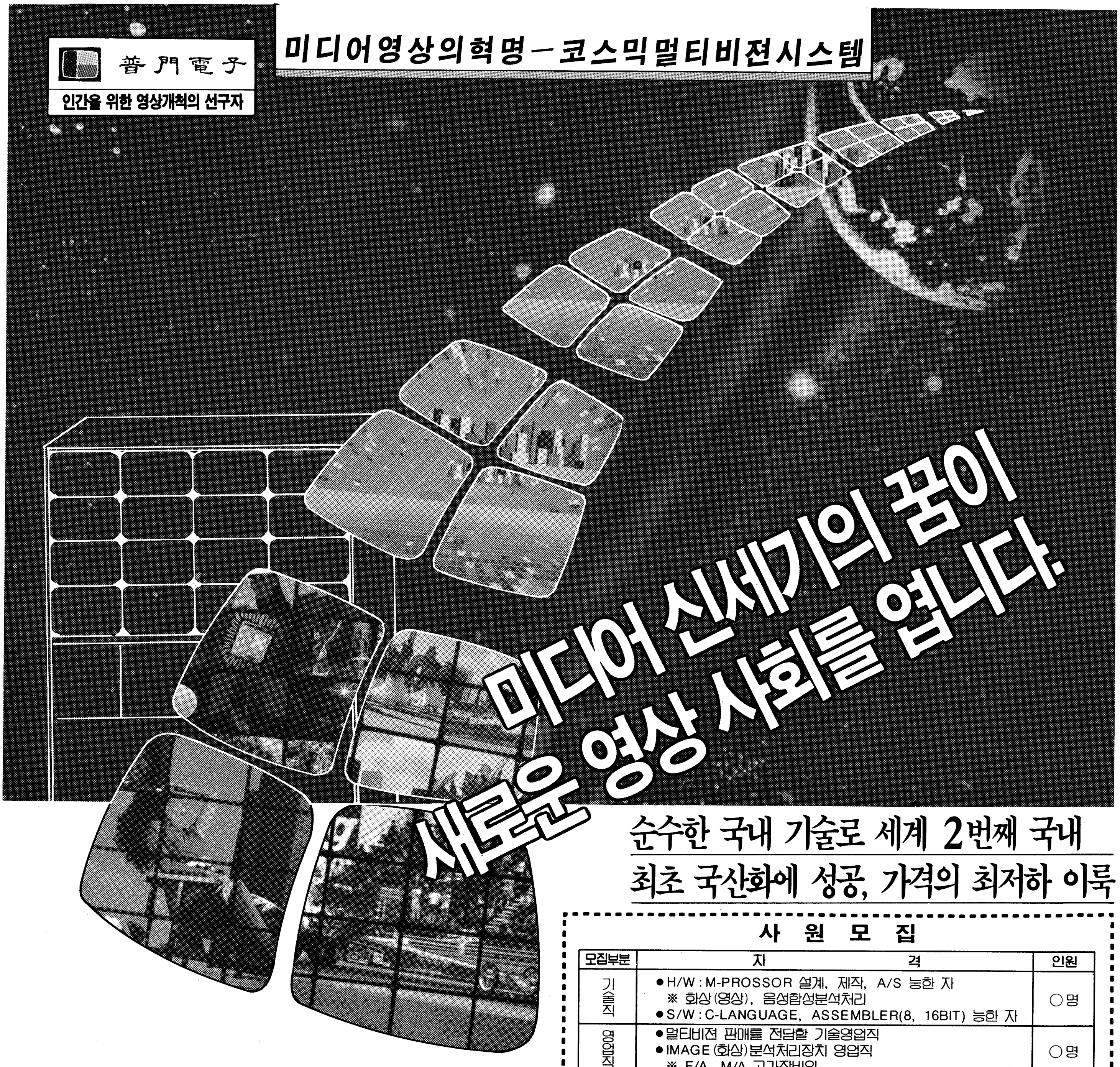
—'89년 5월호 100페이지「마우스 드라이버 툴키트」중에서
다음의 소스 프로그램들이 누락되었음을 알려드립니다.

●Turbo Pascal 버전 툴키트
●Turbo C 버전 툴키트
●Microsoft C 버전 툴키트
●Macro Assembler 버전 툴키트
●마우스를 체크하는 헤더 화일

이상의 소스 프로그램들을 지면관계상 게재하지 못한 점을 사과드리며
이 프로그램들은 디스켓에 수록하여 제공할 예정입니다.

普門電子
인간을 위한 영상개척의 선구자

# 미디어영상의혁명－코스믹멀티비젼시스템




## 순수한 국내 기술로 세계 2번째 국내 최초 국산화에 성공, 가격의 최저하 이룩

### 사 원 모 집

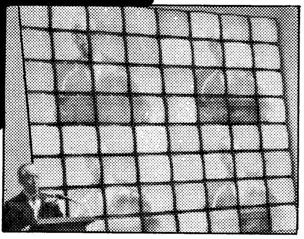
| 모집부문 | 자 격 | 인원 |
|---|---|---|
| 기술직 | ●H/W : M-PROSSOR 설계, 제작, A/S 능한 자<br>※ 화상(영상), 음성합성분석처리<br>●S/W : C-LANGUAGE, ASSEMBLER(8, 16BIT) 능한 자 | ○명 |
| 영업직 | ●멀티비젼 판매를 전담할 기술영업직<br>●IMAGE(화상)분석처리장치 영업직<br>※ F/A, M/A 고가장비임 | ○명 |

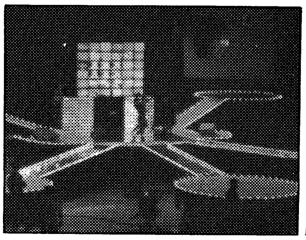
※ 각 부문 공히 경력사원 우대, 책임 운영할 경영자·중견간부 환영
※ 공히 군필자

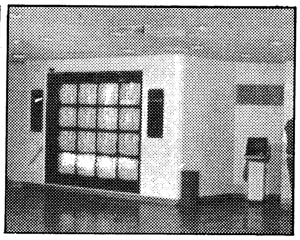
- 제출서류 : 이력서, 자기소개서
- 제출기간 : 공고일부터 (선착순)
- 전형방법 : 서류전형 합격에 의하여 개별면접 및 실시일정 통보
- 제출된 서류는 일체 반환치 않음 (서류는 우편으로 우송 및 연락처 명시)

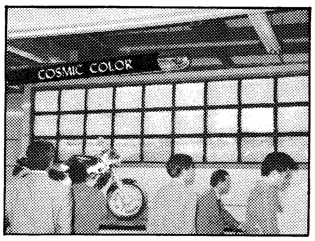
NEW

# 멀티 비젼










■ 코스믹 아트비젼의 활용

●기업홍보 ●전시장 디스플레이 ●방송국콘서트홀 무대장치
●대중집회 행사 ●호텔쇼핑센터, 백화점 ●공공시설 ●각종정보
●고 부가가치 수출품

**새로운 뉴모드 보급형 개발완료 납품준비 임대성업중**

환상의 세계로 !

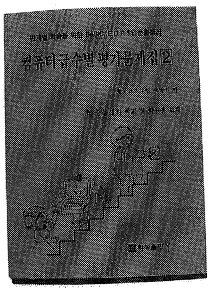
**COSMIC COLOR**
**BOMUN ELECTRONICS CO.,**

576-21, Eungam-Dong, Eunpung-Gu, Seoul, Korea
**TEL : 302-3116, 304-5780**

# 신간안내

## 컴퓨터 급수별 평가문제집 2

한국프로그램개발원/희성출판사 발행
정가 4,000원/228면/4·6배판



과학교육의 중요성이 커짐에 따라 컴퓨터의 조기교육 및 국민학교 교육과정에서도 컴퓨터 교육이 활발해지고 있다.

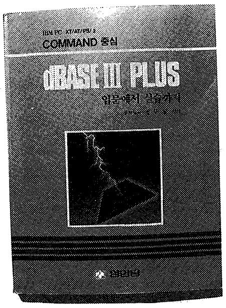
이 책은 그동안의 많은 BASIC 교재와 응용문제집들의 단점을 분석하여 각급 학교의 교육과정과 기타 컴퓨터학원에서 수업용 보조교재로서 활용할 수 있게 꾸며진 책이다.

책 구성은 전체 8급으로 나누어져 있으며, 제8급은 BASIC기본 명령어와 전산기초 개론, 제7급은 기본명령어 숙지와 전산개론, 제6급은 BASIC 일반명령어와 E.D.P.S, 제5급은 BASIC 일반명령어와 프로그램응용, 제4급은 예상문제풀이와 응용, 제3급은 프로그램 기록과 운용, 화일구성과 플로우차트, 제2급은 프로그램 실제작성기법과 운용, 메모리맵과 배열, 제1급은 기본어셈블리어와 ASCII CODE, ROM, RAM구성요건, 도형정의와 BASIC관계, 특수음향과 컴퓨터트릭이 수록되어 있다.

따라서 학생들의 자체실력 평가와 함께 하나씩 등급을 넘어감으로서 자신감을 키워줄 줄 수 있고, 또한 학생자신이 앞으로 배워나가야 할 과제를 분명히 알고 나아갈 수 있도록 구성되어 있으며 제8급, 7급, 6급, 5급 문제와 실시평가문제는 제1권에 실려 있다.

## command중심 dBASE Ⅲ PLUS 입문에서 실습까지

정구철 편저/성안당 발행
정가 8,000원/591면/4·6배판

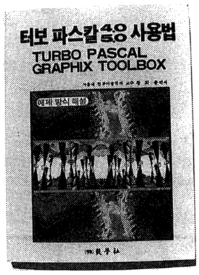



명령어 방식으로 수행하는 dBASE Ⅲ PLUS는 Ashton Tate사의 데이터베이스 관리 패키지로 이 책은 dBASE Ⅲ PLUS를 사용하는 사람들에게 유용한 사전이자 안내서가 될 수 있다. 즉 dBASE 명령어들의 의미와 사용법을 설명과 예를 들어가며 세밀한 사항까지 상세히 설명되고 있음을 볼 수 있다.

명령어와 함수에 관한 대부분의 표제는 구조, 설명 사용법, 주의, dBASE Ⅲ와의 동등어, 관련 해설 등의 항목으로 구성되어 있고 이외에도 50여개 이상의 주요 항목이 dBASE Ⅲ PLUS의 기초와 데이터베이스에 대한 궁금증을 풀어 줄 것이다. 이들 항목들의 내용을 보면 메뉴 운용식 사용자 인터페이스를 사용하며 초보사용자를 위해 설계된 여덟개의 보조 메뉴와 Aliases, Database structure, Fields, Key, Record, Sorting 등 데이터베이스에 대한 필수적인 개념들, 그리고 프로그램을 작성하는데 필요한 테크닉과 툴, 관련 유틸리티, 프로그래밍 환경등이다.

## 터보 파스칼 4.0/5.0 사용법 Turbo Pascal Graphix Toolbox

황희융 편저/교학사 발행
정가 7,000원/457면/4·6배판

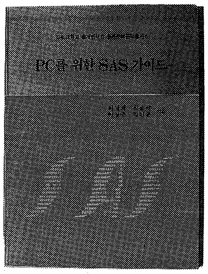



미국 볼랜드사의 간판 소프트웨어라고 할 수 있는 터보 파스칼(버전4.0 또는 버전5.0)은 파스칼 언어뿐 아니라 터보 파스칼상에서 작동되는 몇가지 툴박스들과 유틸리티들이 있다. 이들 중 그래픽 활용도구로써 그래픽스 툴박스(버전4.0)의 사용법을 소개하는 것이 이 책의 주 내용이다. 그래서 터보 파스칼 자체의 사용법을 먼저 알아야 할 독자들의 편의를 위해서 터보 파스칼(버전4.0) 자체의 사용법 및 예제를 간단히 제1장과 제2장에 걸쳐 해설하고 있다. 그리고 터보 파스칼 버전 5.0에서 새로이 추가된 디버깅 기능 사용법 해설이 제3장 및 제4장에 나와있고 제2편의 제0장에는 터보 파스칼 그래픽스 툴박스 전체를 쉽게 개념을 잡을 수 있도록 예제와 함께 해설하고 있으며 제1장 및 제2장은 원래의 이 책의 목적으로서 그래픽스 툴박스의 소개와 예제가 실려있다.

또한 이 책의 많은 분량을 차지하고 있는 제3장에서는 툴박스의 기술적 레퍼런스(Technical Reference)를 다루고 있다. 터보 파스칼을 애용하고 있는 유저에게 좋은 지침서가 되리라 본다.

## PC를 위한 SAS가이드

허명회 외 3인 공저/자유아카데미 발행/정가 3,000원/80면/4·6배판




통계분석에 있어서 컴퓨터 패키지 프로그램은 이제 거의 필수적으

로 사용되고 있다.

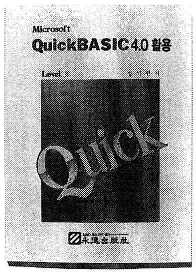
본서는 통계패키지 SAS에 관하여 매뉴얼 형식으로 쓰여져 SAS의 초보자나 유경험자에게 도움이 되도록 실제 SAS작업의 과정이 예와 함께 설명된 SAS입문서이다.

본서의 I부 SAS의 기초편, II부 SAS의 화면 관리시스템, III부 기타 토픽을 통하여 PC를 위한 SAS에 관한 이해를 얻을 수 있을 것이다.

## Quick BASIC 4.0 활용 Level 1

**장석원 저 / 영진출판사 발행**

**정가 5,000원 / 232면 / 4·6배판**




현재까지 발표된 PC용 프로그래밍 언어의 종류는 그 수를 헤아릴 수 없을 만큼 많다. 그중에서도 베이직은 오랜 기간동안 가장 많은 사람들로부터 사랑받아 온 프로그래밍 언어라고 할 수 있을 것이다. 더구나 베이직이 인터프리터라는 고정된 환경을 벗어나 점차 고도화된 프로그래밍 언어로서의 면모를 갖추어 나감에 따라 한층 더 그 사용자의 폭이 넓어지고 있는 실정이다. 그 중에서도 마이크로소프트사의 퀵베이직4.0은 타의 추종을 불허할만큼 강력한 언어로 평가받고 있으며 점차 그 사용자의 폭도 증가하고 있다.

이 책은 초보자가 퀵베이직 4.0을 가장 쉽게 이해할 수 있도록 쓰여졌으며 특히 GWBASIC을 비롯한 인터프리터 베이직을 접해본 사람들이 퀵베이직을 배울 때 겪는 문제점들을 쉽게 해결할 수 있도록 배려했다. 이 책의 주된 내용을 살펴보면 퀵베이직 통합환경의 사용방법과 구조화 프로그래밍의 기법, 그리고 유저 라이브러리의 작성방법과 제작기법, I/O기능의 사용방법, 한글처리 문제, ISAM 라이브러리의 사용법들이 자세하게 실려있다. 그리고 이 책은 "퀵베이직 활용" 시리즈의 첫권으로써 초보자를 대상으로 했으며 이후 Level II에서는 Mixed Programming 의 기법을 비롯한 실무에 관련된 내용이 소개될 예정이다.

## TURBO C 입문과 실습

**한성천 편저 / 크라운출판사 발행**

**정가 7,000원 / 439면 / 4·6배판**




Dennis M. Ritchie에 의해 개발된 이래 C언어는 작으면서도 강력한 기능을 가진 언어로 각광받아 왔다. C가 이와같이 각광받을 수 있었던 것은 무엇보다도 C의 철학인 '프로그래머를 위한 언어' 때문이라고 볼 수 있다. 하지만 이와같은 C의 철학이 C언어를 어렵게 만들어 초보자에게 많은 난해함을 안겨 주었던 것도 사실이다.

이 책은 MS-DOS환경에서 사용할 수 있는 컴파일러중 볼랜드사에서 개발한 터보C 컴파일러를 사용해 쉽게 C언어를 배울 수 있도록 편집하였으며 또한 C언어 전반과 터보 C 모두를 배우는데 많은 도움이 되리라 본다.

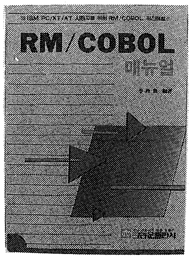
터보 C가 ANSI표준을 따르고 있기 때문에 이 책의 많은 개념들은 다른 컴파일러에도 그대로 적용될 수 있다. 그리고 사용자가 C에 전혀 경험이 없는 프로그래머라는 점을 가정하여 쓰여졌지만 다른 언어(특히 구조화된 언어, PASCAL, MODULAR II 등)로 조금이라도 프로그램을 작성해 본 실무 경력이 있다면 많은 도움이 될 것이다.

책 구성은 제4부로 구성되어 있는데, 제1부는 C언어의 역사에 대해 고찰하고 터보 C의 통합환경과 에디터에 대한 자세한 설명이 나와 있다. 제2부에서는 C언어에 대한 기본적인 사항들에 대해서 나와있고, 제3부는 보다 진보된 주제에 관해, 제4부에서는 Turbo C 1.5에서 추가된 내용을 다루고 있다.

## RM/COBOL 매뉴얼

**이정열 편저 / 크라운출판사 발행**

**정가 6,000원 / 307면 / 4·6배판**

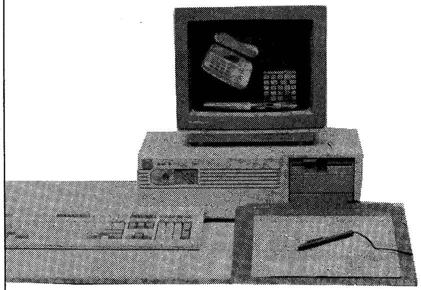



COBOL(Common Business Oriented Language)은 주로 사무분야의 여러 문제를 컴퓨터를 사용하여 해결하기 쉽게 제작된 언어로서, 1960년 CODASYL (Conference on DAta System Language)에 의해 처음 개발된 뒤 몇 차례의 표준화와 개정작업을 통해 발전해 오며 점차 보편화되고 효용성이 높아지게 되었다.

이 책에서는 이러한 COBOL 언어중에서도 사무와 업무분야에서 가장 사용도가 높은 RM/COBOL에 대해서 설명하고 있다. RM/COBOL은 Ryan-Mcfarland 사에서 제작한 컴파일러를 사용하는 COBOL언어(표준 ANS COBOL 1974)로서 IBM-PC 호환기종에서 사용할 수 있으며 표준 운영체제로 MS-DOS나 PC-DOS를 사용하고 있다.

# 신제품

### 그래픽전용 시스템
## Vision-II



본 제품은 그래픽 관련 종사자나 설계관련 종사자, 그리고 창작활동을 하는 아티스트분야에서 활용할 수 있도록 한 다목적 컴퓨터그래픽 시스템이다.

그래픽과 CAD를 동시에 사용할 수 있게 구성한 제품으로서 CAD의 데이터를 그래픽쪽으로 전송, 다양한 색상과 페인팅기법이 요구되는 분야 등에 적합하다. 표현가능한 칼라는 총 32,768가지이다.

해상도면에서는 1인치당 2540라인으로 기존 Summa Sketch에 비해 2.5배 더 정밀하다.

특히 이 시스템에 지원되는 소프트웨어는 Versa CAD와 Tips라는 페인팅 소프트웨어로 구성된 것이 특징이다.

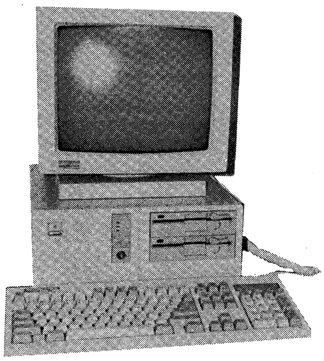
주요 규격은 다음과 같다.

• 하드웨어
–IBM PC / AT 호환
–13" 아날로그 RGB 모니터
–타가 16보드, EMS보드(2M)
–작업영역 : 12"×12"~12"×18"

• 소프트웨어
–Tips(2D 페인팅)
–Versa CAD(2D Drafting, CPL)
–RIO(2D) / TOPAS(3D) 데모화일

문의처 : 어플라이드엔지니어링
전 화 : (02)679—7691

### 12MHz 터보 XT
## 수퍼 88



본 제품은 시스템 클럭속도가 12MHz인 초고속 터보 XT로, 기존에 주류를 이루고 있던 4.77MHz에 비해 2.8배 빠른 속도를 가진다.

또한 메인보드상에서 1MB까지 메모리확장이 가능하고 조합형, 완성형 한글 및 KS 한자를 지원해 주며 NEC의 V20 마이크로프로세서와 5.25인치 FDD 2대 (360KB)가 내장되어 있다.

특히 CGA, MDA, HGA의 기능을 겸비한 멀티비디오 카드를 채택하였다.

주요 규격은 다음과 같다.

• CPU : 16비트 V20(12~15MHz)
• HDD : 옵션
• 시리얼포트 1개, 페러렐포트 1개
• 확장슬롯 : 8개
• 디스플레이 : 14" 그린모니터
  한글 · 한자-40자×25행
  영문-80자×25행
• 키보드 : 101키
• 리얼타임클럭 내장
• 옵션 : 한글카드

문의처 : 마이크로텍스 엔지니어링
전 화 : (02)542—0960

### NEC V20 채택, 16비트 PC
## 로맥스 V2 드림



본 제품은 10MHz의 처리속도를 지닌 저가형 16비트 PC이다.

NEC V20 마이크로 프로세서를 CPU로 채택한 16비트 IBM PC XT 호환기종인 로맥스 V2 드림은 A형과 B형이 있다. 이들의 공통 특징은 주문형 반도체(VLSI)를 사용하여 원가절감을 한 것과 7비트 완성형 한글카드를 내장, 행망용 86키보드 사용 등이다.

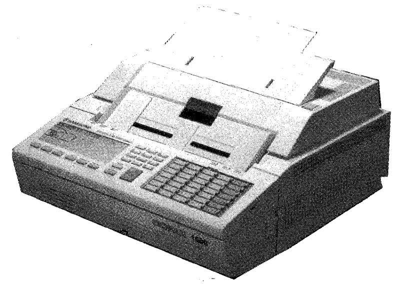
특히 A형은 컴퓨터 본체가 48만원

(부가세 포함)인데 256KB(640KB 확장가능)의 주기억용량을 기본으로 0.5MB FDD 1대가 내장되어 있다. 이 A형의 가격은 기존 저가형 16비트 PC에 비해 7만원정도 저렴한 가격이다.

B형은 주기억용량 640KB에 0.5MB 2FDD를 내장하고 65만원이다.

문의처 : (주)로얄컴퓨터
전 화 : (02)785—3458

## HDP방식 채택 팩시밀리
# COFAX 1600/1600M



본 제품은 수퍼화인(Super Fine)한 정밀도를 실현한 HDP방식을 채용하여 인쇄상태를 고화질로 향상시킨 팩시밀리이다.

이는 삼성의 독자 모드인 HS, EX와 GII, GⅢ 모드를 채택하여 기종에 관계없이 송수신이 가능하고, 열전사방식에 의한 보통용지 기록방식을 채택하였기 때문에 어떠한 필기구의 사용도 가능하며, 영구적인 문서의 보존도 가능하다.

또한 라인상태도 좋지 않아 에러가 발생하는 경우에는 "ECM" 기능을 이용하여 에러가 발생한 부분을 자동으로 재송신해줌으로써 정확한 정보전달이 가능하고, FAX 본체와 전화가 멀리 떨어져 설치된 경우에도 전화버튼 조작만으로 수신이 가능하다.

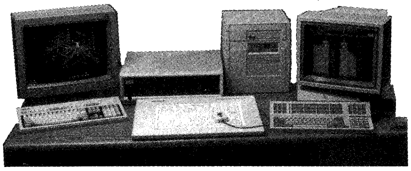
특히 FAX와 전화 사용회선이 공용일 경우 전화를 걸어 온 상대에게 송신방법과 통화방법을 음성으로 안내하여 준후 상대방의 조작순서에 의해 전화를 FAX모드로 자동전환할 수 있다.

주요 규격은 다음과 같다.

- 기억용량 : 2MB
- 유효독취폭 : 최대 296㎜
- 부호화방식 : MH, MR, MMR
- 적용회선 : 가입전화회선, 전용전화회선
- 소비전력 : 대기시-20W, 송신시-48W, 수신시-50W
- 주사선밀도 : (주)8도트 / ㎜, (부) 3.85, 7.7, 15.4라인 / ㎜
- 전송속도 : 9600, 7200, 4800, 2400bps 자동절체

문의처 : (주)삼성애니파
전 화 : (02)678—4618

## 3차원 그래픽 패키지
# 마이크로스테이션32



본 제품은 MS-DOS와 클리퍼 UNIX 상에서 2 / 3차원 Drafting과 디자인 그리고 표면모델링을 하기 위해 운용되는 그래픽 패키지로서, IBM PC XT / AT 및 PS / 2에서 이용이 가능할 뿐 아니라 매킨토시기종에서도 이용이 가능하다. 또 호스트용인 IGDS와도 호환된다.

주로 기계설계나 플랜트 설계 및 토목, 건축설계, 맵핑(Mapping) 분야에서 사용이 가능한 응용소프트웨어이다.

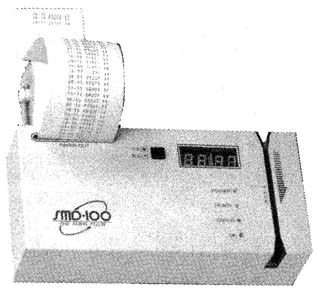
또한 일단 초기단계에서는 적은 비용으로 설치하였다가 필요한 업무의 다양성 정도에 따라 CAE나 CAM까지도 확장이 가능한 것이 장점이다.

주요 규격은 다음과 같다.

- 2 / 3차원 Drafting, 디자인 및 3차원 표면모델링
- 그래픽 데이터베이스 / 논 그래픽 데이터베이스 동시 구축가능
- 사용이 용이한 코맨드
- 다양한 에디팅기능
- 멀티 유저 네트워킹 기능
- 페이퍼 메뉴
- 스크린 사이드 바 / 펑션 / 버튼 메뉴
- 자동적으로 차원 설정
- 타 시스템과의 인터페이스 가능
- 히든라인 / 표면의 처리

문의처 : 인터그래프 코리아(주)
전 화 : (02)784—8725

## 마그네틱 카드리더 시스템
# SMD-100



이 SMD-100은 마그네틱카드로, 출입자가 본 카드를 투입하면 출입시간이 기록되어 프린터로 출력해 볼 수 있는 제품인데 전자식문과 연결된 경우에는 문의 자동개폐 기능도 갖는다.

또한 14자 프린터가 내장되어 있고, 스탠드얼론방식의 독립작동이 가능하며 만년 캘린더로 월, 일, 시, 분을 표시할 수 있다.

특히 RS-232C에 의해 호스트컴퓨터와 링크하여 출입관련 데이터를 호스트로 전송할 수 있는데 TX카드로 300명분의 데이터를 일시에 호스트로 보낼 수 있다.

주로 출퇴근 · 주차장 · 도서관 열람환자 · 회원제 및 창고출입관리 등에 적합하다.

주요 규격은 다음과 같다.

- 기록방식 : ABA
- 통신방식 : RS-232C(1200~9600bps)
- 디스플레이 : 7세그먼트 LED 4디지트
- 프린터 : 14문자 알파벳 수

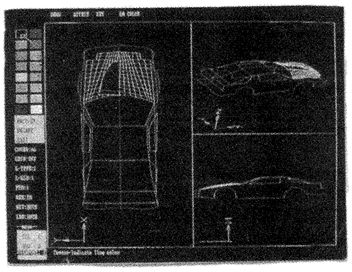
● 게이트 출력 : 24V, 1A(Normal)
● 전원 : AC 110/220V 겸용, 60Hz, 10W
● 외형 : 215×100×55㎜(WDH)

문의처 : 신광자동화시스템(주)
전 화 : (02)468-6945

또 모든 명령이 아이콘메뉴(Icon-menu) 방식을 취하고 있어 컴퓨터에 대한 개념이 없는 초심자도 설계 능력만 있으면 쉽게 사용할 수 있다.

문의처 : 제우스컴퓨터
전 화 : (02)784-7841

## 아이콘 메뉴방식 채택한 CAD패키지 MAXXICAD



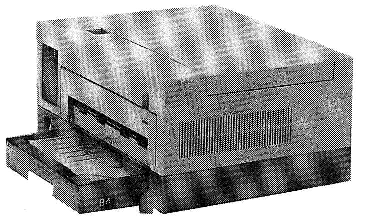
본 제품은 IBM PC/XT/AT/386 기종 및 호환기종에서 사용이 가능한 CAD 패키지로 3차원을 요구하는 기계, 금형 설계 등에 적합하다.

이 MAXXICAD는 3차원 표현이 가능하기 때문에 설계자가 화면을 4등분으로 나누어 정면도, 측면도, 평면도를 한 눈에 보면서 설계작업을 할 수 있고 나머지 한 부분에서는 정면도, 측면도, 평면도를 연결한 같은 용적의 3차원 상태를 볼 수 있다.

또한 4개로 나누어진 화면중 어느 한부분에서 설계작업이 이루어지면 나머지 3화면도 동시에 이루어지므로 설계자가 의도한 내용을 3차원으로 파악할 수 있고, 3차원으로 만든 오브젝트를 X, Y, Z축으로 회전시킬 수 있어 오브젝트의 전체면을 모두 볼 수 있다.

더우기 설계하는 과정에서 14개의 윈도우를 사용할 수 있어 부분적으로 확대·축소하면서 작업할 수 있으며, 히든라인 제거, 명암작업과 3차원 설계를 한 모듈안에서 이용할 수 있어 사용이 간편하다. 특히 명암작업을 와이어 프레임 모델링상태에서 하기 때문에 표면처리까지 가능하다.

## 제로그래피방식의 고화질 레이저프린터 XP-11



이 XP-11은 제록스 고유의 레이저 제로그래피(Xerography)방식과 1성분 현상방식을 채택하여 화질을 대폭 개선한 소형경량의 데스크탑형 레이저프린터이다.

그리고 A4의 경우 매분 11매, B4의 경우는 9배의 고속출력이 가능하고 최대 B4 사이즈에서 최소엽서사이즈까지 다양한 용지사이즈로 프린트할 수 있다.

또 Face-Up과 Face-Down 2가지 용지배출방식을 채용하여 Face-Up은 250매까지, Face-Down은 200매까지 적재할 수 있는데 Face-Down 출력의 경우는 페이지순으로 프린트아웃되므로 페이지바꿈을 하는 시간이 절약되고 토너와 드럼을 카트리지방식으로 채용하였다. 총 250매의 대량서류를 연속프린트하는 것도 가능하다.

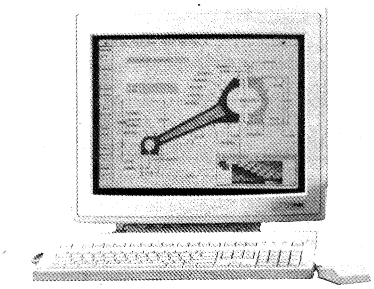
특히 EPSON 도트 매트릭스 프린터에 대응하는 표준모드에 의한 각종 PC 및 대형, 미니 마이크로 컴퓨터와 호환이 가능하고 한글·한자 폰트가 풍부하며 풀그래픽에 의한 임의의 직선, 사선, 곡선 처리기능과 10단계 세도우 패턴기능 등을 갖고 있다.

주요 규격은 다음과 같다.

● 한글 : 24×24, 32×32, 명조 고딕
● 영문 : 24×32, 16×32, 전ASCII 문자
● 한자 : 24×24, 32×32 명조 4,888자 (옵션)
● 해상도 : 240DPI
● 메모리사이즈 : 1.5MB
● 최대전력 : 0.7kW
● 크기 및 무게 : 478×389×270㎜ (WDH)/25kg

문의처 : 코리아 제록스(주)
전 화 : (02)757-7903

## 다기능 워크스테이션 매킨토시 Ⅱx



본 제품은 엔지니어링분야는 물론 그래픽, 전자출판, 사무자동화에 이르기까지 다방면의 업무를 한꺼번에 지원하는 워크스테이션이다.

이 매킨토시 Ⅱx는 한글·한자를 완벽하게 구현하는 한글 OS Talk™외에 유닉스의 애플버전인 A/UX, IBM-PC의 MS-DOS까지 지원하는 등 다양한 OS 사용이 가능하다.

그리고 건축 CAD나 디자인과 드래프팅, AI시스템, PCB드로잉 등과 같은 다양한 엔지니어링분야의 솔루션들이 개발되어 있어 사용자·목적에 따라 편리하게 선택하여 사용할 수 있다.

또한 1,680만가지의 칼라 팔레트에서 256칼라를 동시 표현할 수 있고, 256 그레이레벨 표시도 가능하다.

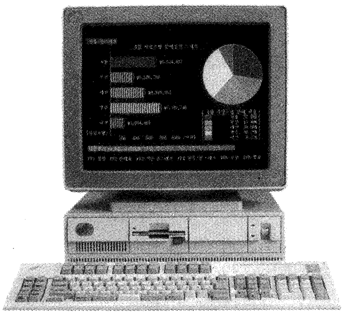
특히 내장된 Local Talk™ 카드를 이용하여 매킨토시 패밀리간의 연결은 물론 IBM-PC와도 LAN을 구성할 수 있다.

주요 규격은 다음과 같다.

• 프로세서 : 68030(15.667MHz),PMMU(Paged Memory Management Unit)를 기본으로 내장
• 코프로세서 : 68882
• 메모리 : 기본4MB, 온보드에서 8MB, 슬롯이용시 128KB까지 확장가능
• ROM : 256KB, SIMM방식에 의한 설계로 확장 · 수정 용이
• FDD : 3.5" 내장형(1.44MB)-400K-B, 800KB 디스켓 리드/라이트/포맷 가능
• HDD : 80MB를 기본으로 160MB, 300MB 부착가능
• 인터페이스 : 6개의 Nu-버스 슬롯, 2개의 RS-422/432 포트, SCSI 페러럴포트, 사운드포트, 2개의 애플 데스크탑 버스
• 인터페이스 전송속도 : 230.4K baud/sec
• 비디오 디스플레이 : 애플 비디오보드
• 키보드 : 애플 확장키보드(105키)
• 마우스 : 1버튼(90펄스/인치)

문의처 : 엘렉스컴퓨터
전 화 : (02)780—4545

## 멀티태스킹이 가능한 업무용 컴퓨터
# PS/5550-T



본 제품은 고속 32비트 인텔 i80386 (20MHz) 마이크로프로세서를 채택하고 최고 16MB의 기억용량과 한글처리 능력을 가진 IBM PS/55 패밀리의 모델 5550-T이다.

특히 IBM의 최신 마이크로채널 아키텍처를 채택하여 멀티태스킹 처리를 지향, 더욱 빠르고 원활한 시스템 처리능력을 가진 것이 특징이다.

주기억용량은 표준 2MB이나 시스템보드 상에서 8MB까지 확장이 가능하며 32비트 선택슬롯을 사용하면 최대 16MB까지도 확장이 가능하다. 여기서 16MB는 시스템보드상의 기억메모리 용량을 포함한 것이다.

그리고 기술계산용의 고속수치연산을 위해 80387(20MHz) 수치연산 공용프로세서도 장착가능하다.

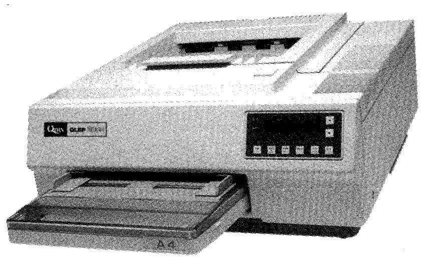
그래픽모드는 1024×768 도트의 고해상도로 262, 144색중 16색 또는 256색을 동시에 표시할 수 있다.

주요 규격은 다음과 같다.

• FDD : 3.5"(1.44MB) 1개
• HDD : 30MB(5551-TK9), 60MB (5551-TKA), 120MB(5551-TKB)
• 비디오 램 : 1MB(2화면분)
• 표준인터페이스 : 비동기통신 1포트 및 마우스 1포트
• 확장선택 슬롯수 : 32/16비트용 2개, 16비트용 1개
• 소요전력(KVA/W) : 0.18/130(평균), 0.32/230(최대)
• 기타 확장선택기구 및 액세서리 : 2MB/4MB 시스템보드, 기억확장키트, 3270 통신접속기, 비동기 TTL접속기, 토큰링 네트워크 접속기, IEEE488 접속기 등

문의처 : 한국 아이·비·엠(주)
전 화 : (02)781—5550

## 데스크탑 레이저 프린터
# QLB3000



본 제품은 12MHz MC68000을 CPU로 채용한 고속 레이저 빔 프린터로 HP 레이저제트 모드를 100% 지원하도록 되어 있어 레이저 빔 프린터를 사용하는 각종 영문 그래픽 소프트웨어의 출력속도도 향상되어 있다.

또한 LCD 프론트 패널(Front Panel)을 채택하여 종전의 딥스위치방식보다 조작성이 개선되어 메시지의 이해와 사용이 편리해졌고 시리얼/페러럴을 동시에 채용하고 전력소모를 줄인 OA에 적합한 기기이다.

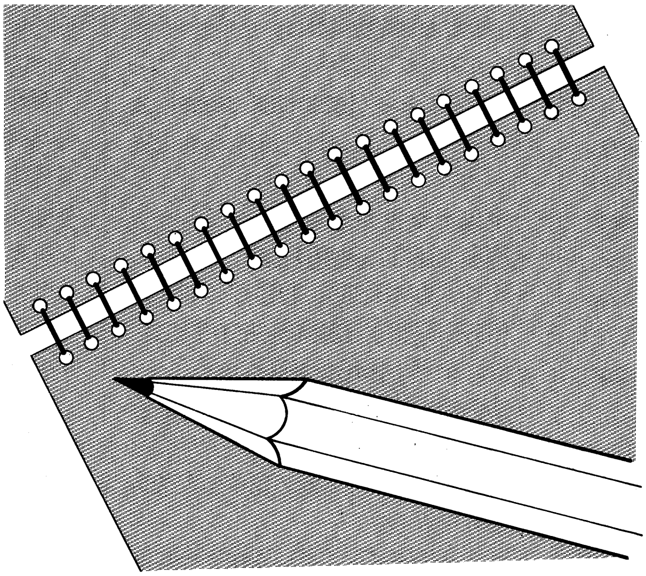
특히 국가 표준코드를 사용하고 있어 행정전산망용 표준 소프트웨어들도 지원할 수 있도록 설계되어 있는 것이 특징이다.

주요 규격은 다음과 같다.

• 인자방식 : 반도체 레이저 건식전자 사진방식
• 인자속도 : 분당 최대 8매
　　첫장 인자 18초
　　동작준비시간 2분이내
• 해상도 : 300DPI
• 사용문자 : 한글(명조, 고딕)-모든 한글, 한자(명조), 영문(Roman Italic)- ASCII
• 그래픽기능 : 풀그래픽, 직선, 사선, 박스, 원호 등
• 칼라 : 카트리지교환방식 단색(흑색, 암갈색)
• 인터페이스 : RS-232C
• 전원 : 115V, 60Hz, 최대 810W(인자시), 최대 95W(대기시)
• 크기 및 중량 : 460×470×218mm(WDH) /23kg

문의처 : (주)큐닉스
전 화 : (02)783—4301

# 플로피디스크



플로피디스크는 기존의 매체에 비해 정보 보존면에 있어 그 어느 것보다도 손쉽게 다룰 수 있는 장점이외에도 디스크를 자켓에 보관하여, 시스템이 있는 곳이라면 어디나 가져가 보존된 정보를 꺼내 볼 수 있을 뿐만 아니라 새로운 정보를 입력할 수 있는 특징을 가지고 있다.

이번 달에는 이러한 정보기록 매체중 일반 유저에게 가장 보편적으로 사용되고 있는 플로피디스크에 대해서 알아보고자 한다.

물론, 처음 의도는 국내에서 생산되는 플로피디스크를 생산 업체별로 살펴보고, 수입대리점 등도 망라하여 독자로 하여금 플로피디스크 구입에 도움을 주고 싶은게 편집부의 의도였다.

이런 취지아래 가능한 "에러율" 등도 체크하고 국내생산업체의 시장점유율, 선호하고 있는 수입품 등도 살펴볼 거창한(?) 의도였지만 국내에서는 몇 안되는 업체가 생산하고 있으며 각 업체가 내놓은 자료의 신빙성도 의심스럽고, 에러율 체크도 몇개의 샘플로 확인한다는 한계 등에 부딪쳐 유저가 쉽게 접하는 플로피디스크중에 도대체 어떤 것들이 우리 주위에 있으며 이를 잘 활용할 수 있는 방법을 모색해보는 것이 실질적인 도움이 되리라 느껴 이런 취지로 플로피디스크를 소개하고자 한다.

우리는 정보의 보관을 위해서 기록매체를 이용하곤 한다. PC에서 기록매체라 하면 크게는 플로피디스크와 하드디스크 등을 들 수 있고, 앞에서도 말했듯이 이 플로피디스크의 구조, 특성, 종류 등을 알아보고 제조공정 등도 살펴보겠다. 플로피디스크란 말이 알려지기 시작한 것은 IBM사에서 처음 자사의 제품에 8인치 디스크를 사용하면서 부터였다. 그후 미국의 슈가트사가 5.25인치 디스크드라이브를 발표하여 5.25인치 디스켓이 확산되기 시작하였으며, 현재 PC에서는 대부분이 5.25인치 디스켓을 주로 사용하고 있다.

하지만 80년도에 들어와서 일본의 소니사에 의해 개발된 3.5인치 디스켓이 발표되고 이를 PC제조업체에서 기억매체로 선택함에 따라 급격히 플로피디스크의 시장을 석권하고 있다 (더욱 자세한 내용은 '본지 2월호 특집' 부분을 참조하기 바람).

## 디스크의 구조

5.25인치 디스크의 구조를 그림 1과 같이 간단하게 나타내었다. 여기에 나타난 기능 등을 살펴보면 다음과 같다.

**① 자켓**

이물질로부터 자기디스크를 보호하며, 자켓 안쪽에는 라이너가 부착되어 있어 정전기 방지, 디스크의 손상방지, 먼지제거 등의 역할을 한다.

**② 인덱스 홀**

신호 및 정보기록 시작신호를 감지하는 홈이며, 디스크가 회전하는 동안 두개의 홀(자켓, 자기디스크)이 완전히 일치 되었을 때 드라이브의 센서가 홀을 감지하여 디스크 표면에 기록을 시작한다.

**③ 헤드윈도우 (Head Window)**

디스크의 표면을 노출시켜 R/W헤드와 접촉하여 읽기/쓰기 기능을 수행하기 위한 장방형 노출부이다.

**④ 중앙홀(Center Hole)**

드라이브가 센터홀을 통하여 자기디스크를 안전하게 죄어 회전시킬 수 있는 구멍이다.

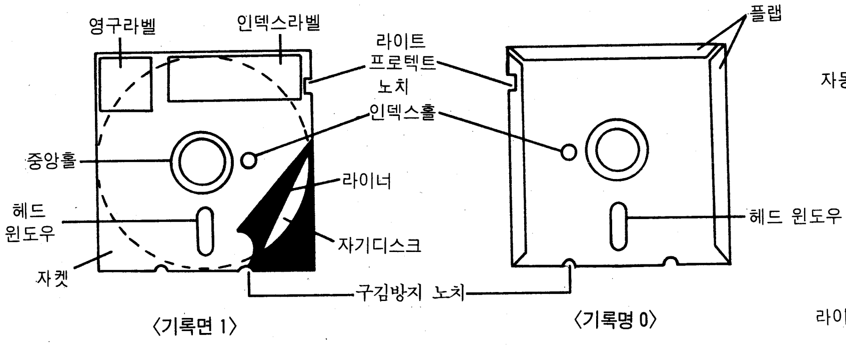
**⑤ 라이트 프로텍트 홀**

우발적인 사고로 인한 데이터 삭제를 방지하기 위하여 사용되는 홀이다(이 부분을 막을 경우 디스크에 기록되지 않는다).

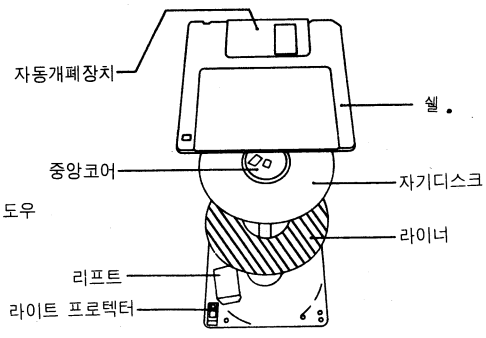
**⑥ 자기 디스크**

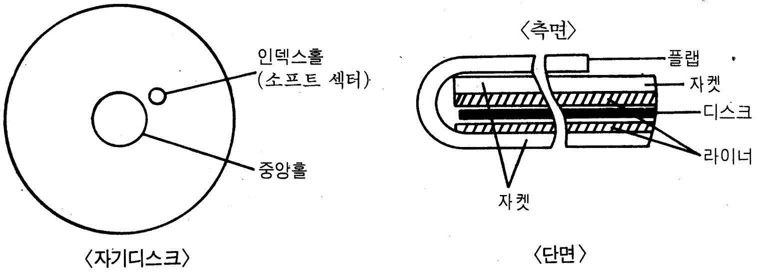
데이터가 기록되는 부분이고 소재는 PET(Polyethylene Terephalate)로 되어 있다. 양면에 산화철과 바인더가 함께 도포되어 있다. 소재의 두께는 75$\mu$m이고 산화물 코팅 두께는 2.5$\mu$m (4 8TPI, 96TPI), 1.5$\mu$m(2HD) 정도이다.

〔그림 1〕5.25" 플로피디스크의 구조




〔그림 2〕3.5" 플로피디스크의 구조




이 밖에도 최근에 각 기종들에 부착되어 나오고 있는 3.5인치 디스켓의 주요기능은 다음과 같다(그림 2 참조).

① 자동 개폐장치

드라이브에 삽입할 때는 자동적으로 열리고, 평상시에는 닫혀있어 외부오염으로부터 디스크를 보호하는 장치이다.

② 쉘(SHELL)

외부의 충격으로부터 자기디스크를 보호하는 장치이며, 5.25인치의 자켓과 같은 역할을 담당한다.

③ 중앙홀

메디어의 균일한 회전을 유도하는 기능을 한다.

④ 라이트 프로텍트

우발적인 데이터 입력으로 인한 기존의 데이터 손실을 방지하는 장치이다(아래로 내릴 경우 기록이 되지 않음).

⑤ 라이너

자기디스크 상의 오염물질을 크리닝하는 기능을 한다.

영구라벨에 붙어 있는 등급구분을 살펴보면 다음과 같다.

**1D**:단면양밀도

**2D**:양면양밀도

**1DD**:단면양밀도양트랙

**2DD**:양면양밀도양트랙

**2HD**:양면고밀도양트랙

**256HD**:FACOM용으로 포맷된 2HD

또한, 영구라벨에 표기된 MD(5.25인치의 경우)는 미니-플로피디스켓을 가리키며, 3.5인치에서는 MF(마이크로-플로피디스켓임을 가리킴)로 나타낸다. 이것은 8인치 디스켓이 표준디스켓으로 채택된 이후에 개발된 제품에 따라 부르는 이름이다. 한편 5.25인치의 경우 어떤 제품은 HG(High Grade)를 표시하고 있는데, 이는 그 회사에서 만든 제품의 상표내지는 품질의 우수함을 소비자에게 인식시키려는 의도이다.

우리가 가장 많이 사용하고 있는 것은 2D나 2HD를 들 수 있는데, 이들의 제품은 디스크 제조공정에 기초가 되는 베이스 필름과 자화물질의 분포밀도 등에 기인된 것으로 보통 정보의 기억용량에 차이를 두고 있다.

## 디스크의 특성

디스크는 사이드 1면과 0면으로 되어 있으며 자켓에 라벨이 붙은 면을 사이드 1, 반대면을 사이드 0이라 한다. 사이드 0, 1에는 48TPI(Track Per Inch) 경우 최외주에서 최내주 쪽으로 트랙00~트랙39까지 40개의 동심원 기록부(트랙)가 있다. (96TPI 는 트랙00~트랙79까지 80개의 트랙이 있다). 각 트랙은 다시 섹터로 나뉘며, 섹터는 다시 ID 필드와 데이터 필드로 나누어 진다. 그리고 ID 필드에는 섹터를 식별하기 위한 어드레스 정보가, 데이터 필드에는 사용자의 입력 데이터가 기록되며, 용량은 하드웨어에 의해서 결정된다.

① 포맷이란

최초의 플로피디스크는 진공상태인 관계로 데이터를 기록하기 위해서는 미리 디스켓에 식별정보를 넣어둘 필요가 있다. 이와같이 디스크의 섹터를 구분하여 데이터를 읽고, 쓰기 편리하도록 ID 필드와 데이터 필드로 나누어 구획정리를 하는 것을 디스크 포맷이라 한다. 플로피 디스크는 포맷된 다음에 비로소 데이터의 입출력이 가능해 진다.

포맷 방법은 모두 알고 있다고 생각되며, 혹시 궁금한 독자(?)는 본지 88년 12월호, 89년 1, 2월호를 보면 잘 설명되어 있다.

② 플로피디스크 포맷구조

디스크의 포맷구조는 그림 3에 나타내었다.

**트랙**:데이터를 기록하기 위한 동심원의 기록띠이고, 40개, 80개의 두종류가 있다.

**실린더**:사이드 0과 사이드 1의 동일번호 트랙의 쌍을 나타낸다.

**섹터**:1트랙을 컴퓨터의 기능에 따라 여러 부분으로 나눈 것으로 섹터는 ID필드와 데이터 필드로 구성된다.

**ID 필드**:트랙, 섹터 신호 정보 등이 기록된다.

**데이터 필드:** 사용자의 데이터를 기록하기 위한 명령으로, 데이터 필드에 입력가능한 기록용량을 레코드의 길이라고 말하며 128, 256, 512, 1024 바이트 등 4종류를 가진다.

**예비 트랙:** 디스크에서 결함 트랙이 생겼을 경우에 대비하여 존재하는 여분의 트랙으로 보통 최내각의 2개 트랙을 지칭한다.

**갭:** 디스크 상의 데이터가 기억되어 있는 트랙과 트랙사이에 있는 공간(데이터 보호를 위해 트랙 상에 설치된 일정공간)

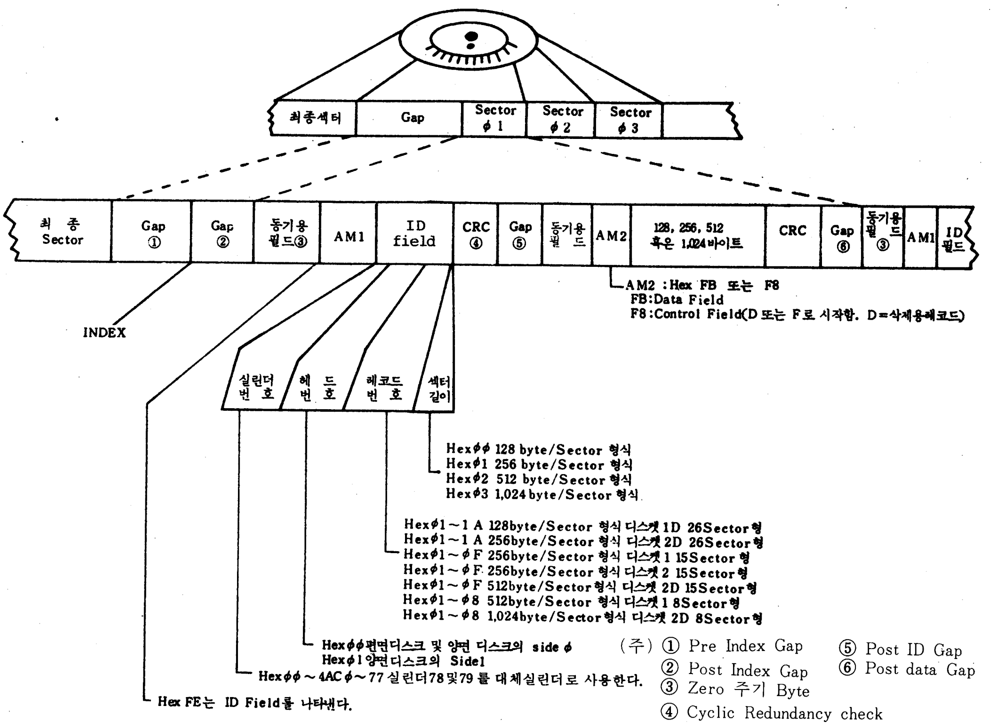
## 기록방식

플로피디스크에 정보를 기록하는 방식은 디스크 드라이브에게 내리는 명령에 따라 FM(Frequency Modulation)방식과 MFM(Modified Frequency Modulation)방식의 2가지가 있다. 단밀도인 경우는 FM방식으로 기록하고, 배밀도인 경우는 MFM방식으로 기록된다. 플로피디스크에 기록하기 위해서 데이터비트의 정확한 위치에 기록하기 위한 클럭비트(Clock Bit)라는 것이 필요하다. 클럭비트와 데이터비트 모두 전기적인 신호로 디스크 드라이브에 전달된다. 데이터비트는 일정한 간격으로 기록되어야 하는데 이 간격을 비트셀(Bit Cell)이라고 한다.

인코딩후에 합성된 데이터 클럭신호는 증폭되어 리드/라이트 헤드코일에 공급한다. 이때 헤드에 전달된 펄스는 자극을 인코딩 펄스에 따라 배열함으로서 디스크에 정보를 기억시킨다.

이와같이 문자를 기억시키기 위해서 인코딩한 것은 미세한 자극의 패턴으로 디스크 트랙위

〔그림 3〕 플로피디스크 포맷구조(5.25″)




에 형성된다. 또한 리드 펄스는 리드/라이트 헤드가 통과하는 면의 자극변화가 있을 때마다 발생한다.

FM 기록방식은 그림 4와 같이 비트셀의 중간에서 데이터비트를 기록한다. 비트셀은 클럭비트에 의해서 결정되며 비트셀이 시작되는 시점이 클럭비트가 전달될 때이다.

MFM 기록방식은 클럭비트를 기록하는 방식이 FM방식과는 다르다. 다시말하면 매번 클럭비트를 기록할 필요를 없앤 것이다. 즉 기록해야 할 비트수를 줄일 수 있고, 비트 셀도 반으로 줄어든 것이다.

이와같이 디스크에서 사용되는 기록방식은 FM과 MFM방식으로 되어 있고, MFM방식은 FM방식에 비해 기록밀도가 2배나 되며, 고밀도 디스크에 기록할 때 사용된다.

## 기록밀도

디스크의 기록밀도를 나타내는 단위에는 BPI(Bit Per Inch)와 TPI(Track Per Inch) 등 2가지로 나타낸다.

BPI는 트랙의 선에 따라 1인치당 기록할 수 있는 데이터의 비트수 이다. 그리고 TPI는 인치당 존재하는 트랙의 수이다. 5.25인치는 기록할 수 있는 TPI가 48TPI와 96TPI이다.

## 제조공정

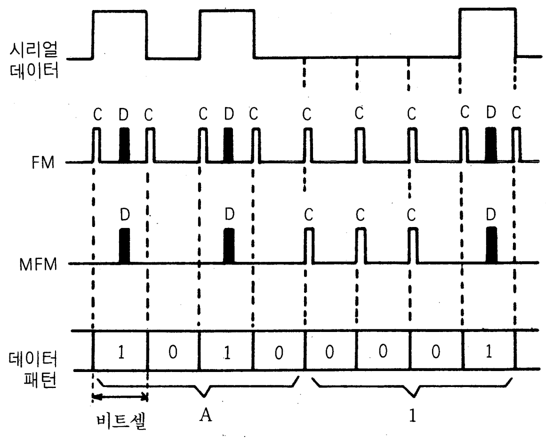
디스크가 어떻게 만들어지는지 과정별로 간략하게 알아보겠다(그림 5 참조). 플로피디스크 제조공정은 비디오 테이프나 카세트 테이프의 공정과 유사하지만 제품의 형태에서 차이가 있다. 그리고 실제적으로 플로피디스크 제조기법은 회사마다 감추고 있어 공개하길 꺼리고 있는 실정이다.

디스크 제조과정에 있어 원단을 수입하여 조립하는 회사도 있고, 아예 자켓(라이너도 포함)에 매체를 넣어 봉하고 테스트를 하여 출하하는 곳도 있다.

이런 상황은 생산원가를 감안하여 택하는 경우로 볼 수도 있고, 매체를 감싸고 있는 라이너가 고속회전에서도 매체에는 손상을 주지않으면서 묻은 먼지를 제거해야 하고, 자켓이 디스크의 품질을 좌우하고 있어 이러한 제조과정의 어려움 때문에 수입에 의존하는 실정이다.

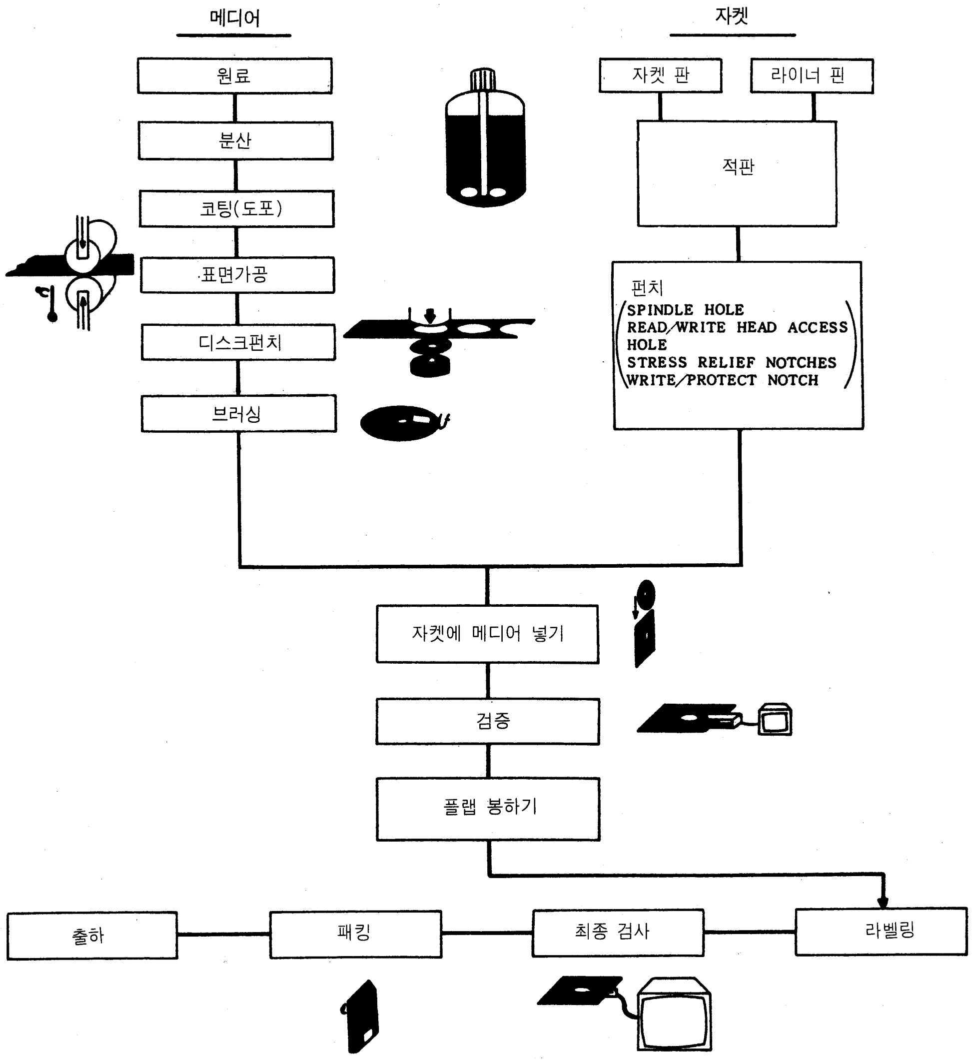
그림 5에서와 같은 과정을 거쳐 디스크가 만들어지며, 원료로는 베이스 필름, 자성체, 수지, 용제, 윤활제, 대전방지제 등이 있으며 이를 일정비율로 혼합하여 베이스 필름에 코팅한다. 또한 코팅된 필름은 표면가공을 결쳐 테이프의 표면조도 및 충진성을 좋게 한다. 이런 과정을

〔그림 4〕 FM, MFM 기록의 비트패턴(A1인 경우)




데이터를 기록하기 위해서 필요한 길이를 비트셀이라 한다.

〔그림 5〕 플로피디스크 제조공정




통해 플로피디스크가 만들어지며 검사과정을 거쳐 완제품으로 공급된다.

벌크디스크와 라벨을 붙쳐 시판되는 일반 디스크의 제조공정에서는 별다른 차이가 없지만 벌크디스크를 구입하여 사용한 예가 있다면 어느 정도의 신뢰도를 갖는 예도 있을테고, 귀중한 데이터를 혹시나 잃어버리면 어쩌나 하는 우려로 사용을 기피하는 사람도 있겠지만 이의 선택은 독자들이 경우에 따라, 선택했으면 한다.

참고로 벌크디스크를 구입에 있어 값싸다고 왕창 구입하기 보다는 한, 두통을 구입해 테스트로 사용해보든지 주위에서 같은 제품을 써본 사람의 반응을 듣고 구입하는 것도 알뜰한 방법이라 생각된다. 그리고 플로피디스크 값이 점점 떨어지는 추세에 있으므로 급히 쓸 일이 아니고서는 사용 용도이외에 카피용으로 한, 두통 정도 예비로 보관하면 편리하게 쓸 수 있다.

## 디스크 활용

디스크 활용에 있어 우선적으로 권하고 싶은 것은 중요한 데이터를 하드디스크에 보관하기 보다는 꼭 플로피디스크 백업본을 만들어 두면, 하드디스크가 깨지는 경우나 타인이 시스템을 사용하다 지워버리는 등의 사고로부터 보호할 수 있어 필요할 때 요긴하게 사용할 수 있다.

이때 가능한 백업본은 디스크 자켓의 라이트 보호 노치를 막아 기록을 방지하면 더욱 편리하다.

또한 유저가 갖고 있는 유틸리티를 이용하여 디스크를 효율적으로 다룰 수 있는 방법이 주위에 많으므로 이를 구입해 활용하면 효과적으로 디스크를 사용할 수 있다.

예를 들어 일반적으로 가장 많이 사용하고 있는 NORTON, PCTOOLS, 사이드킥 등의 유틸리티를 들 수 있고, 화일을 압축하여 저장하는 아카이브, AT에서 360K를 800K로 포맷하여 사용할 수 있는 KFAMAT.COM, 800K.COM 등을 들 수 있다.

이외에도 디스켓을 효과적으로 사용, 관리하는 유틸리티가 많이 있으며 본지에서는 이를 계속 발굴하여 독자에게 소개하

고 있다.

요즘들어 특히 3.5인치 FDD를 장착하고 있는 시스템이 많이 나오고 있어 3.5인치 디스크도 앞으로 국내에서 많이 사용되리라 예상된다. 3.5인치 디스크는 5.25인치에 비해 크기가 작으면서 많은 정보를 저장할 수 있으며, 자켓이 쉘이라는 플라스틱으로 싸여 매체가 노출되어 있지 않아 보관이 용이하다.

여러분의 시스템에 3.5인치, 5.25인치 FDD를 장착하고 있다면, 보관하는데 우선적인 것은 3.5인치 디스크에 저장해 두는게 훨씬 편리하다.

간혹 디스크를 시스템이 읽지 못하는 경우가 있는데 이는 시스템 자체의 문제도 있지만 디스크 사용에 있어 유저가 범하는 실수가 대부분이다. 이런 문제는 "주의사항"에서 다루겠다.

## 수입품과 벌크

앞에서도 벌크에 대해서는 약간 언급했지만 현재 국내에 수입되는 디스켓의 많은 종류가 여기에 속하며 국내에서도 이를 생산하고 있다.

벌크는 생산된 제품의 성능이 불안할 때, 회사에서 확실히 신뢰할 수 없는 경우에 이를 싼가격으로 유출한 것이었지만 이제는 버젓이 벌크디스켓을 하나의 상품(!)으로 내놓고 있다.

또한 제품의 다양화란 명목(?)으로 자회사의 고유상표가 있는데도 이를 표기하지 않은채 마치 다른 회사의 제품인양 혹은 벌크의 모습으로 둔갑하여 고유 브랜드는 원래의 가격으로 판매하고, 이보다 싼가격을 찾는 소비자를 위해 이를 공급하고 있는 실정이다.

이는 지금처럼 디스켓이 흔하지 않을 때, 외국으로부터 수입한 디스켓을 사용하는 처지와는 달리 의도적으로 저가격의 상품을 국내에 보급하고 있다. 가격차 때문에 예전에 많이 사용하는 MAXWELL, DATALIFE, MEMOREX 등의 제품은 구입하기도 어렵고, 국내 생산되는 제품의 품질이 좋아 국산품을 선호하기 때문에 자취를 감췄지만 대신 일본, 대만, 홍콩산 벌크디스켓을 보따리 장사꾼(?)이 시세에 따라 마구잡이로 들여오고 있어 종류도 다양하고 금방 단종되기도 한다. 이런 연유로 이들 제품을 구입한 후에 이상이 있으면 교환받을 여지가 없어 신중을 기하는게 좋다.

외국산 벌크디스크와 더불어 국내 생산업체에서도 이와 유사한 제품을 시중에 내놓고 있는 상황이니 이제는 벌크를 불량품으로 보기보다는 하나의 아이템(?)으로 받아 들여 테스트를 해보고 싼값으로 구입하는게 최선이라 하겠다.

## 주의사항

플로피디스크는 데이터를 기억하고 있는 중요한 매체임에도 불구하고 그 중요성을 인식하지 못하고 푸대접하는 경우가 있다. 그렇게 취급할 경우 후회할 일이 꼭 발생하고야 만다.

플로피디스크의 취급시 주의사항은 보통 겉 자켓에 그림이나 글로 나타나 있어 디스켓을 사용할 때마다 접하게 되므로 디스켓 취급시 주의사항에 대해서는 모두다 알고 있으리라 본다.

간혹 XT에서 사용하고 있는 디스크를 가지고 AT에서 정보를 받아 다시 XT에서 나머지 작업을 하려 할 때 시스템이 이를 읽지 못하는 메시지가 나타난다. 이는 XT와 AT의 액세스 타임(호출시간이라고도 함. 메모리나 주변기기에 대해서 데이터번지를 지정한 다음에 그 데이터가 실제로 얻어지기까지의 시간 또는 전송이 개시되기까지에 소요되는 시간)에 기인하기 때문이다.

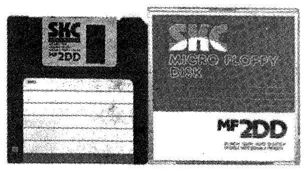
마지막으로 플로피디스크를 취급하는데 유념할 것은 디스크 자체를 살아있는 생명체로 생각하고 다룬다면 문제는 없을 것이다. PC

# (주)SKC

▼ 5.25인치



▼ 3.5인치




## ● 5.25인치 플로피디스크

| 모델명 | 2D | 2DD | 2HD | HIGH-GRADE | | |
|---|---|---|---|---|---|---|
| | | | Formatted | 2D | 2DD | 2HD |
| 기록면 | 양면 | 양면 | 양면 | 양면 | 양면 | 양면 |
| 기록밀도 | 배밀도 | 배밀도 | 고밀도 | 배밀도 | 배밀도 | 고밀도 |
| 트랙수 | 40×2 | 80×2 | 80×2 | 40×2 | 80×2 | 80×2 |
| 최대기록밀도 (BPI) | 5,876 | 5,992 | 9,650 | 5,876 | 5,922 | 9,650 |
| 기록용량 (Unformatted) | 500KB | 1.0MB | 1.2MB | 500KB | 1MB | 1.6MB |
| 기록방식 | MFM | MFM | MFM | MFM | MFM | MFM |
| 항자력 (Öe) | 300 | 300 | 600 | 300 | 300 | 600 |
| 트랙밀도 (TPI) | 48 | 96 | 96 | 48 | 96 | 96 |

## ● 3.5인치 플로피디스크

| 모델명 | 1DD | 2DD | 2HD |
|---|---|---|---|
| 기록면 | 단면 | 양면 | 양면 |
| 기록밀도 | 배밀도 | 배밀도 | 고밀도 |
| 트랙수 | 80 | 80×2 | 80×2 |
| 최대기록밀도 (BPI) | 8,190 | 8,190 | 13,947 |
| 기록용량 (Unformatted) | 500KB | 1MB | 2MB |
| 기록방식 | MFM | MFM | MFM |
| 항자력 (Oe) | 600 | 600 | 730 |
| 트랙밀도 (TPI) | 135 | 135 | 135 |

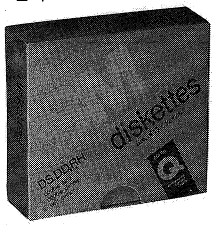
● 제조원 : (주)SKC ● 전 화 : (02) 756-5151

●대리점 :

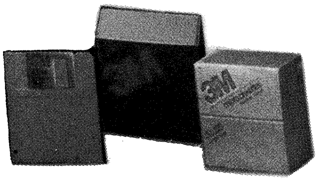
| | | | |
|---|---|---|---|
| (서울) 아프로만 | 545-8813 | (부산) 대성전산 | 805-2285 |
| 맥산컴퓨터 | 273-6835 | (광주) 유노우컴퓨터 | 27-4119 |
| 바스콤 | 234-1918 | (대전) 한국명상전자 | 622-7357 |
| 진전산 | 234-4500 | (인천) 선경전산 | 763-6288 |
| 오림전산 | 744-2085~7 | (대구) 대원컴퓨터랜드 | 60-1801 |
| 석영전산 | 272-6811~6 | 아프로만 | 253-1712 |
| 원영실업 | 249-4547~9 | (마산) 한길컴퓨터 | 94-5555 |
| 상일상사 | 272-3338 | (대전) 대양 O.A | 253-5455 |
| (부산) 캐드콤비지네스 | 806-9870 | (수원) 동양신소재 | 37-1234 |
| (대구) 세림전산 | 22-7474 | (광주) 유일종합상사 | 367-2711 |

## 한국쓰리엠(주)

▼ 5.25인치




▼ 3.5인치



### ●5.25인치 플로피디스크

| 모 델 명 | SD | DD | HD |
|---|---|---|---|
| 기 록 면 | 단면 | 양면 | 양면 |
| 트랙수/면 | 35 | 40/80 | 80 |
| 트랙밀도(TPI) | 48 | 48/96 | 96 |
| 최대기록밀도(BPI) | 2.7K | 5.8K/5.9K | 9.8K |
| 기억용량(Unformatted) | 109K | 500K/1M | 1.6M |
| 전 송 속 도 | 250K | 250K | 250K |
| 항자력(Öe) | 300 | 300 | 675 |

### ●3.5인치 플로피디스크

| 모 델 명 | STD | HD |
|---|---|---|
| 기 록 면 | 양 면 | 양 면 |
| 트 랙 수 / 면 | 80 | 80 |
| 트랙밀도 (TPI) | 135 | 135 |
| 최대기록밀도 (BPI) | 8.7K | 17.4K |
| 기억용량 (Unformatted) | 1.0M | 2.0M |
| 전 송 속 도 | 250K/500K | 500K |
| 항 자 력 (Öe) | 665 | 720 |

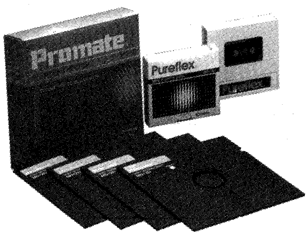
●제조원 : 한국쓰리엠주식회사　●전 화 : (02) 785-4567

●대리점 :

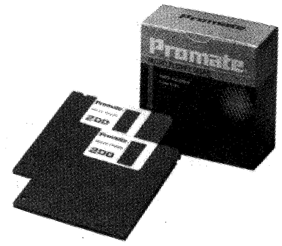
| | | | |
|---|---|---|---|
| (서울) 태광실업 | 274-0778 | (주) 토피아 | 532-4177 |
| 소프트월드 | 712-5331 | 태양산업(주) | 555-8967 |
| 삼양비지네스포옴 | 718-2349 | 성신미술공사 | 752-3984 |

## 삼우전자

▼ 5.25인치




▼ 3.5인치




### ●5.25인치 플로피디스크

| 모 델 명 | MD2D | MD1DD | MD2DD | MD2HD | MD-2HD 256 |
|---|---|---|---|---|---|
| 기 록 면 | 양면 | 단면 | 양면 | 양 면 | |
| 기 록 밀 도 | 배 밀 도 | | | 고 밀 도 | |
| 트 랙 수 | 40×2 | 80 | 80×2 | 80×2 | 80×2 |
| 트랙밀도 (TPI) | 48 | 96 | 96 | 96 | 96 |
| 최대기록밀도 (BPI) | 5876 | 5576 | 5922 | 9650 | 9650 |
| 기억용량 (Unformatted) | 500KB | 500KB | 1MB | 1.6MB | 985088 (포맷시) |
| 기 록 방 식 | MFM | MFM | MFM | MFM | MFM |
| 전 송 속 도 (bit/sec) | 250K | 250K | 250K | 500K | 500K |
| 항자력 (Öe) | 300 | 300 | 300 | 600 | 600 |

### ●3.5인치 플로피디스크

| 모 델 명 | MF1D | MF2D | MF1DD | MF2DD | MF2HD |
|---|---|---|---|---|---|
| 기 록 면 | 단 면 | 양 면 | 단 면 | 양 면 | |
| 기 록 밀 도 | 배 밀 도 | | | | 고밀도 |
| 트 랙 수 | 40 | 40×2 | 80 | 80×2 | 80×2 |
| 트랙밀도 (TPI) | 67.5 | 67.5 | 135 | 135 | 135 |
| 최대기록밀도 (BPI) | 8128 | 8650 | 8190 | 8720 | 13947 |
| 기억용량 (unformatted) | 250KB | 500KB | 500KB | 1MB | 1.6MB |
| 기 록 방 식 | MFM | MFM | MFM | MFM | MFM |
| 전 송 속 도 (bit/sec) | 250K | 250K | 250K | 500K | 500K |
| 항자력 (Öe) | 650 | 650 | 650 | 650 | 650 |

●제조원 : 삼우전자　●전 화 : (02) 236-1764

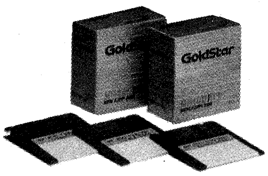
●대리점 :

| | | | |
|---|---|---|---|
| (서울) 신즈써프라이 | 778-3505 | (부산) 신성전산 | 462-5993 |
| 소프트빌리지 | 540-1621 | 하이콤컴퓨터 | 861-7473 |

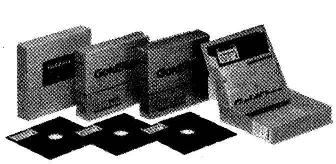
(대구) 제일전산 69-1289
삼보컴퓨터상사 422-8422
(광주) 일진컴퓨터 27-8086
대흥데이타 364-3157
미성컴퓨터 68-3072
(대전) 제일컴퓨터 253-8266
프린스컴퓨터 255-3364
(창원) 창원컴퓨터 83-0808
(울산) 울산컴퓨터 93-9991~5
(제주) 데이타베이스 52-4480

## 금성사

▼ 3.5인치




▼ 5.25인치



### ●5.25인치 플로피디스크

| 모델명 | 1D | 2D | 2DD | 2HD |
|---|---|---|---|---|
| 기록면 | 단면 | 양면 | 양면 | 양면 |
| 기록밀도 | 배밀도 | 배밀도 | 배밀도 | 고밀도 |
| 트랙밀도(TPI) | 48 | 48 | 96 | 96 |
| 최대기록밀도(BPI) | 5536 | 5876 | 5922 | 9650 |
| 기록용량 | 290KB | 500KB | 1MB | 1.6MB |

### ●3.5인치 플로피디스크

| 모델명 | 1DD | 2DD | 2HD |
|---|---|---|---|
| 기록면 | 단면 | 양면 | 양면 |
| 기록밀도 | 배밀도 | 배밀도 | 고밀도 |
| 트랙밀도(TPI) | 135 | 135 | 135 |
| 최대기록밀도(BPI) | 8190 | 8720 | 17434 |
| 기록용량 | 500KB | 1MB | 1MB |

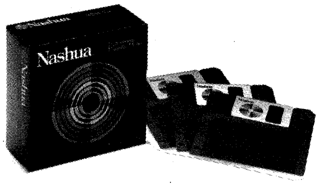
●제조원 : 금성사
●전 화 : (02) 757-1234
●대리점 :
(서울) 금성사무기 900-0900~5
서울비지네스 234-2121
한양전산 336-1411
성인상사 568-0353
토피아 590-4678
(부산) 홍진상사 22-1212
(대구) 고려전산 423-0722
(광주) (주) 동화 525-4111
(대전) 성인상사 823-3905

## 나슈아

### ●5.25인치 플로피디스크



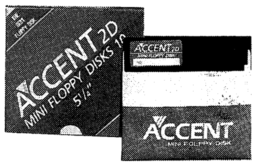

### ●3.5인치 플로피디스크 (2D, 2HD)




●공급처 : 소이상사
●전 화 : (02) 551-5631~5
●대리점 :
(서울) 대하전산 777-5361
소프트빌리지 545-6313
성도컴퓨터 554-4484
한서기업 540-5787
(부산) 부산컴퓨터 808-3662
(마산) 한이컴퓨터 43-6386
(청주) 국제전산 54-4884
(대전) 영컴시스템 522-9317

## 액센트

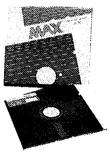
### ●5.25인치 플로피디스크 (2D)




●공급처 : (주) 아프로만
●전 화 : (02) 545-8813

## MAX

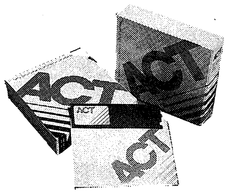
### ●5.25인치 플로피디스크 (2D)




●공급처 : (주) 아프로만
●전 화 : (02) 545-8813

## ACT

### ●5.25인치 플로피디스크 (2D,2HD)




●공급처 : 에이스컴퓨터
●전 화 : (02) 272-8800

보내는 사람

□□□-□□□

우편요금
수취인후납부담
발송유효기간
1989. 2. 1～1991. 12. 31
서울 체신청
승인119호

월간 퍼스널컴퓨터 하이테크

서울특별시 용산구 갈월동 17

퍼스널컴퓨터 편집부앞

140-150

월간 퍼스널컴퓨터의 자매기관



| | | | | | | |
|---|---|---|---|---|---|---|
| 애독자페이지 | 관심있는 분야 | | 게재를 원하는 기사 | | | |
| | ☐ 컴퓨터에 관한 알고 싶은 질문 | | | | | |
| | ☐ 본지에 바라는 의견 | | | | | |
| 정기구독신청서 | 구독신청방법 | ☐ 은행지로 | ☐ 우체국 대체구좌 | ☐ 소액환 | ☐ 기타 | |
| | 구독기간 | 년 월호부터 | 년 월호까지 | | 구독신청일자 | |
| | ☐ 받으실 분의 이름과 주소(우편번호) ☎ ( — ) | | | | | |

※ 이곳에 풀칠을 하여 뒷면 수신란이 나오게 접어서 우체통에 넣어 주세요(우표부착 불필요).

# 애독자 설문지

책의 편집에 반영하고자 하오니 성의껏 적어서 보내주시기 바랍니다.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 작성자 | 이름 | | 나이 | | 직장 | | 성별 | |
| | 주소 | ☎ ( — ) | | | | | | |
| 사용기종 | 8비트 | | 16비트 | | 32비트 | | | |
| | 주변기기 | □ 모니터(컬러, 단색) □ 드라이브(HDD, FDD) □ 프린터(9핀, 24핀, 레이저) □ 인터페이스카드( ) | | | | | | |
| | 사용용도 | □ 취미 □ 업무 □ 교육 □ 학습 □ 기타 | | | 사용경력 | | | |
| | 사용언어 | □ 베이직 □ C □ 파스칼 □ 기계어 □ 기타 | | | | | | |
| | 사용분야 | | | 사용 소프트웨어 | | | | |
| 책구입방법 | 구입방법 | □ 정기구독 □ 서점구입( 서점) □ 증정 □ 기타 | | | | | | |
| | 구입동기 | □ 정기구독 □ 필요한 내용이 있어서 □ 주위에서 권해서 □ 기타 | | | | | | |
| | 본지외 구독잡지 | □ 없다 □ 있다(잡지명 : ) | | 원하는 발행일자 | | | | |
| | 애로사항 | □ 가격이 비싸다 □ 종이 질이 나쁘다 □ 인쇄상태가 나쁘다 □ 기타 | | | | | | |
| | □ 본지 구독을 추천하고 싶은 사람이 있다면(이름과 주소) ☎ ( — ) | | | | | | | |
| 이번달에 좋았던 기사(2가지) | 제목 | | 제목 | | | | | |
| | □ 이유 | | □ 이유 | | | | | |
| 이번달에 싫었던 기사(2가지) | 제목 | | 제목 | | | | | |
| | □ 이유 | | □ 이유 | | | | | |

자르는 선

# 월간 퍼스널컴퓨터의 정기구독자에게는 아래와 같은 특전이 마련되어 있습니다.

—정기구독자는 본사가 보유한 각종 정보 및 자료를 무료로 배부받으실 수 있읍니다.
—본지에 게재되는 각종 프로그램을 디스켓에 담아 무료로 보내드립니다.
—본지 및 본지의 자매지(월간 전자과학) 주최로 개최되는 세미나 및 각종 행사에 무료로 참석하실 수 있읍니다.
—본지에서 행하는 각종 부대사업에 할인혜택이 주어집니다.
—구독기간중 책 가격이 인상되더라도 종전가격으로 구독이 가능합니다.

## ●구독신청은

—시중은행에서 지로용지를 이용하시거나 우체국에서 대체구좌 또는 소액환으로 송금하시면 됩니다.
은행 지로번호 : 3000667
우체국 대체구좌번호 : 010256—31—0526772

## ●정기구독료는

—1 년 구독시 30,000원
—6개월 구독시 15,000원

## ●구독신청시 주의사항은

—구독기간과 주소(우편번호), 전화번호와 받으실 분의 성명을 정확히 기입하시기 바랍니다.
—배달사고나 기타 변동사항은 본지 영업부로 즉시 연락하시기 바랍니다. 신고전의 분실사고는 책임지지 않습니다.

## ●정기구독 문의처는

—서울시 용산구 갈월동 17 ㊜ 140-150
월간 퍼스널컴퓨터 영업부
전 화 : (02)754—1044(대표)
FAX : 757—1049

## ▨ 전국총판 ▨

| | | | |
|---|---|---|---|
| ●서울● | 종로, 중구, 마포, 용산 | 명 문 사 | 774—6314 |
| | 중랑, 성동, 동대문 | 홍 익 도 서 | 454—1810 |
| | 노원, 성북, 도봉 | 진 원 사 | 996—2443 |
| | 강남, 강동, 송파 | 광 일 사 | 415—7330 |
| | 관악, 동작 | 관 악 사 | 889—4504 |
| | 서대문, 은평 | 형 제 사 | 304—9462 |
| | 영등포, 구로 | 영 도 사 | 613—5381 |
| | 강서, 양천 | 연 우 사 | 865—8239 |
| | | 진 명 서 점 | 272—5871 |
| | | 관 음 서 적 | 266—7792 |
| ●부산● | | 종합기술서적 | 462—2431 |
| | | 영광도서전시관 | 805—9500 |
| | | 북 부 총 판 | 805—9677 |
| | | 남 부 총 판 | 756—2727 |
| ●대구● | | 청 운 서 림 | 23—7335 |
| | | 한 일 서 적 | 23—7101 |
| | | 학 원 사 | 22—0336 |
| ●인천● | | 동 양 서 점 | 73—3384 |
| ●광주● | | 삼 복 서 점 | 22—0258 |
| ●경기● | 부천 | 대 경 서 점 | 62—3672 |
| | 안양, 안산 | 사 랑 글 방 | 57—1916 |
| | 수원 | 동 아 서 점 | 5—6436 |
| | 성남 | 영 광 서 점 | 2—6532 |
| ●강원● | 원주 | 원주종로서적 | 42—2943 |
| | 삼척 | 영 동 사 | 72—6060 |
| | 강릉 | 삼 일 사 | 2—2314 |
| | 춘천 | 대 진 서 점 | 54—5148 |
| ●충북● | 충주 | 문 학 사 | 2—2814 |
| | 청주 | 화 문 당 | 55—0503 |
| ●충남● | 천안 | 동방서적공사 | 63—3314 |
| | 대전 | 동 양 서 점 | 22—1538 |
| ●경북● | 포항 | 경 북 서 점 | 2—2717 |
| | 구미 | 중 앙 서 점 | 52—7915 |
| ●경남● | 진주 | 동 아 서 점 | 42—3768 |
| | 진해 | 학 애 서 점 | 2—2523 |
| | 울산 | 우 정 서 점 | 43—6003 |
| | 마산 | 기 술 서 적 | 56—1417 |
| ●전북● | 전주 | 홍 지 서 점 | 6—5311 |
| | 군산 | 동 원 서 점 | 42—3776 |
| ●전남● | 목포 | 문 화 당 | 5—3086 |
| | 여수 | 대 양 서 점 | 62—2111 |
| ●제주● | | 광 양 서 적 | 53—5077 |

# 이달부터 프로그램 디스켓 제공방법이 바뀌게 됩니다.

프로그램 디스켓 제1집은 1989년 6월1일부터 6월30일 현재 정기구독자에 한하여는 무료로 보내드립니다.

**컴퓨터의 보급이 늘어남에 따라 이제는 어디에서나 쉽게 접할 수 있을 만큼 대중화되었으며 컴퓨터란 용어자체를 자연스럽게 대화속에 끌어들일 수 있게 되었습니다. 그러나 아직까지는 우리가 활용할 수 있는 소프트웨어가 부족하다고 할 수 있습니다. 저희 퍼스널컴퓨터에서는 이러한 분들을 위하여 매달 책에 게재되는 프로그램을 디스켓에 담아 제공하고자 합니다.**

## □ 프로그램 제공안내

이번 달부터 프로그램 디스켓 제공방법이 바뀌게 됩니다. 그동안 정기구독자에 한하여 직접 본사로 오실 경우 무료로 복사해 드렸으나 많은 사람들이 몰려 업무에 지장을 초래하는 바 이번 달부터는 직접 찾아오시는 정기구독자에 한해서는 디스켓 1장당 1,500원에, 지방에 계신 분은 종전과 같이 디스켓 구입가격(1장당 1,500원)+송료(자켓 포함 1,000원), 그리고 일반 독자에게는 디스켓 1장당 4,000원씩에 제공할 예정입니다.

**● 정기 구독자의 경우**

— 직접 찾아오시면 본사 영업부 디스켓 담당직원이 정기구독자인지를 확인(주민등록증이나 기타 증명서 필요) 한후 프로그램 디스켓 1장당 1,500원씩에 판매하게 됩니다.

— 지방에 계신 분은 본지 뒷면에 있는 은행지로 용지를 이용하여 필요하신 프로그램 디스켓 명을 적으신 후 디스켓 구입비(1장당 1,500원, 1장을 초과할 경우 1장당 1,500원씩 증가함)와 송료(자켓 포함해서 2장단위로 1,000원씩 증가)를 본사 영업부 프로그램 디스켓 담당자 앞으로 보내주시면 됩니다.

**● 일반 독자의 경우**

— 일반 독자의 경우 프로그램 디스켓 1장당 4,000원씩에 판매하게 되며 제공방법은 정기 구독자와 같습니다.

## □ 문의처

주소 : 서울시 용산구 갈월동 17 세기빌딩 3층
월간 퍼스널컴퓨터 영업부 디스켓 담당자앞
전화 : (02) 754-1044
FAX : (02) 757-1049

# 프로그램 디스켓

## ■ 제 1집

**디스크 논리구조를 이용한 화일복구**(88년 12월호 p74)
플로피디스크상에서 실수로 지워진 화일들을 완벽하게 복구시켜 준다.

**LX-800의 버그**(89년 3월호 p160)
LX-800에서 그래픽을 하드카피할때 생기는 버그를 제거해준다.

**스피디 3의 제작**(89년 2월호 p166, 3월호 p192)
마이크로 로보트의 소스 프로그램

**360KB 이상의 데이터를 복사하려면**(89년 4월호 p54)
360KB 이상의 데이터를 분리하여 저장할 수 있게 해준다.

**레코드판 관리 프로그램**(89년 4월호 p94)
200개의 곡명을 수록할 수 있으며 원하는 형태별로 찾아볼 수도 있다.

**미분이 가능한 계산기 프로그램**(89년 4월호 p110)
공학에서 자주 사용되는 복잡한 계산을 처리할 뿐만 아니라 미분도 가능하다.

**그래픽 애니메이터**(89년 4월호 p176)
20개의 프레임에 연속적인 이미지를 그린 다음 빠른 속도로 출력시켜 움직임을 표현한다.

**QUICKEY.COM**(89년 4월호 p200)
램상주 유틸리티로서 키보드의 감도를 빠르게 해준다.

**교육용 프로그램**(89년 5월호 p76)
물리학습용 프로그램으로 교육용 프로그램의 구성은 어떻게 되어 있나를 알 수 있다.

**전화번호부 관리 프로그램**(89년 5월호 p88)
디베이스로 간단히 작성한 프로그램으로 전화번호와 주소를 입력하여 검색하거나 우편용 레이블 등을 작성할 수 있다.

□ **주의**
각 화일들은 아카이브화 되어 있기 때문에 확장시킬 때 용량이 모자랄 수도 있으므로 이점에 유의하여야 하며 프로그램을 실행시키기전에 반드시 README화일을 불러 확인한다.

# 유틸리티

## ■ 마우스 활용 유틸리티

**마우스 드라이버 툴키트**(89년 5월호 p94)
마우스 드라이버 프로그램으로 각자가 손쉽게 자기용도에 맞게 응용할 수 있는 프로그램 툴이다.

**그래픽 에디터 SEM-DRAW**(89년 5월호 p112)
여러가지 그래픽에 필요한 루틴들을 첨가하여 마우스를 이용해 손쉽게 할 수 있도록 한 에디터이다.

# 공개 소프트웨어

## ■ PC매거진(89년 3월호 p200)
KEY-FAKE.COM을 비롯한 41개의 유용한 유틸리티가 담겨 있다.

## ■ DX.COM(89년 5월호 p198)
디렉토리 관리 기능을 가진 DX.COM 등 여러가지 다양한 프로그램들이 수록되어 있다.

# 하이테크 퍼스널컴퓨터

과월호의 구입은 책속에 있는 은행지로용지를 이용하시거나
대체구좌 또는 소액환을 이용해
필요하신 호수를 기입하신후 본사로 신청하시면 됩니다.
송료는 본사가 부담합니다.




□ 88년 12월호 주요목차
- 초보자 페이지 [1] 어떻게 시작하나요 ?
- 테크닉 디스크 논리구조를 이용한 화일복구/베이직과 어셈블리 루틴을 결합한다
- 특집 IBM PC 철저해부
  [1] 호환성이란 무엇인가/[2] XT의 하드웨어/[3] AT의 하드웨어/[4] IBM PC의 국내현황
- 활용 VDISK의 제작/하드디스크의 셋업/배치화일을 만들어 보자
- 유틸리티 소개 [1] 노턴 유틸리티
- 제작코너 애플컴퓨터를 이용한 조명제어장치
- 구입가이드 IBM-PC/XT/AT/386




□ 89년 1월호 주요목차
- 초보자 페이지 [2] 화일의 초기화와 백업
- 테크닉 DOS의 기능호출/디스크 드라이브의 응급처치/터보 파스칼의 강력한 기능들
- 특집 IBM PC ROM BIOS의 분석
  IBM-PC의 BIOS/중요한 BIOS들/BIOS 사용상의 유의점/MS-DOS의 특징
- 활용 에러처리 루틴의 구성/하드디스크의 설치/메모리 확장
- H/W 리뷰 만능보호장치 HASP-II
- 제작코너 음성출력시계/전2중 1200bps용 카드형 모뎀
- 구입가이드 인터페이스 카드




□ 89년 2월호 주요목차
- 초보자 페이지 어떻게 시작하나요 [3] 화일관리
  둘리가 본 컴퓨터세계 [1] 컴퓨터 길들이기
  MS-DOS 입문 [1] IBM-PC의 첫걸음
- 테크닉 C 언어에서의 최적화
- 기획 허큘리스 그래픽카드
- 특집 디스크 드라이브의 구조 [1] 하드 디스크 드라이브
  [2] 플로피 디스크 드라이브
- 활용 배치화일로 키의 변환을/비디오램의 속성변환
- 제작코너 마이크로 로보트/스피디 3의 제작 (하드웨어편)
- 유틸리티 소개 [2] 노턴 코맨더
- 구입가이드 프린터

# 과월호 구입안내

□ **은행지로번호** : 3000667
□ **우체국 대체구좌번호** : 010256-31-0526772
□ **가 격** : 3,000원(권당)
□ **문의처** : 월간 퍼스널컴퓨터 영업부
전화 : (02)754—1044, FAX : (02)757—1049




**□ 89년 3월호 주요목차**

- 초보자 페이지 MS-DOS 입문 ② **디렉토리 구조**
  둘리가 본 컴퓨터세계 ② **컴퓨터 그는 누구인가?**
  어떻게 시작하나요 ④ **배치화일**
- 테크닉 **LX-800의 버그**
- 공개 소프트웨어 소개 **PC 매거진**
- S/W 리뷰 **마이크로 캡(II)**
- 특집 **MS-DOS의 구조** 화일시스템/내부구조/프로그램의 실행
- 활용 **배치화일로 인스톨 작업을/키보드의 스캔코드 값을 구한다**
- 제작코너 **마이크로 로보트/스피디 3의 제작(소프트웨어편)**
- 유틸리티 소개 ③ **아카이브**
- 구입가이드 **디스크 드라이브**




**□ 89년 4월호 주요목차**

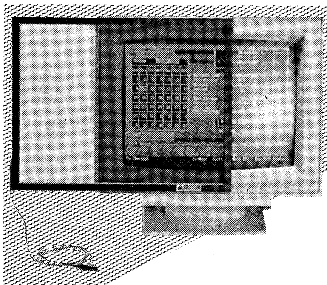
- 초보자 페이지 MS-DOS 입문 ③ **COMMAND.COM과 외부명령**
  둘리가 본 컴퓨터세계 ③ **컴퓨터와 선보는 요령**
  어떻게 시작하나요 ⑤ **입·출력 I/O의 재지정**
- 활용 **화일의 결합과 분리/간단한 방법으로 키정의를 바꾼다/그래픽 애니메이터/메모리할당과 할당해제**
- 테크닉 **360KB 이상의 데이터를 복사하려면/미분이 가능한 계산기 프로그램/QUICKEY.COM의 분석**
- 특집 **PC통신이란 무엇인가** RS-232C란/접속형태/케이블/통신속도/통신방식
- H/W 리뷰 **POWER 88**
- S/W 리뷰 **DASOFT-16**
- 구입가이드 **모니터**




**□ 89년 5월호 주요목차**

- 초보자 페이지 어떻게 시작하나요 ⑥ **파이핑과 필터**
  유용한 유틸리티 ① **PC Tools**
  둘리가 본 컴퓨터세계 ④ **컴퓨터와 컴퓨터 사이**
- 활용 **디렉토리의 결합과 분리/하드디스크에서의 파킹/파스칼에서의 지수계산/TGSHIP.COM의 분석**
- 테크닉 **마우스 드라이버 툴키트/마우스를 이용한 그래픽에디터 SEM-DRAW**
- 제작코너 **사인보드**
- 특집 **MS-DOS.SYS의 분석**
- 워드프로세서 **발송문 관리**
- 공개 소프트웨어 소개 **디렉토리에 관한 모든 기능을 가진 DX.COM**
- 구입가이드 **모뎀**

# 애독자 선물



3명

폴리에스터에 특수 구리도금과 아크릴 수지로 가공 처리된 첨단기술의 V.D.T 증후군 방지용 컴퓨터 보안기이다.

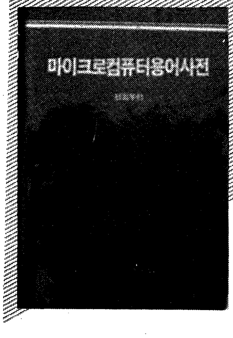
삼종실업 제공
TEL : 784 - 6931




5명

컴퓨터에 전문지식이 없는 사람도 쉽게 접근할 수 있도록 꾸며진 좋은 지침서이다.

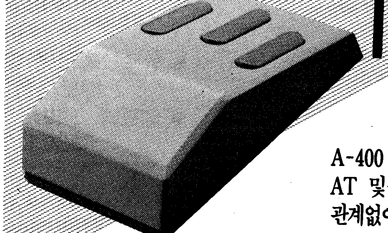
영진출판사 제공
TEL : 719 - 1119




5명

누구나가 한권씩은 지니고 있어야 하는 컴퓨터 용어사전으로 비교적 자세히 설명되어 있다.

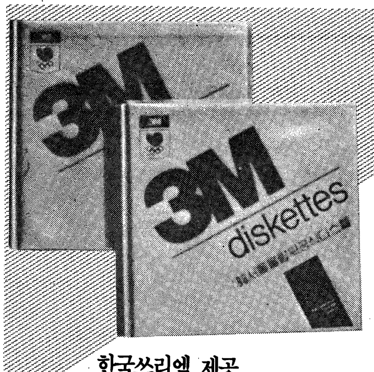
오므사 제공
TEL : 754 - 1044



1명

A-400 마우스는 IBM PC/XT/AT 및 호환기종이면 기종에 관계없이 사용이 가능하다.

하나컴퓨터 제공
TEL : 717 - 4894




3명

특수 산화물 코딩 처리된 에러율 0%의 디스켓이다.

한국쓰리엠 제공
TEL : 785 - 4567




5명

미국 UL규격을 인정받은 제품으로 아크릴 시트로 특수가공되어 선명한 화면을 유지시켜준다.

화신전자 제공
TEL : 733 - 3567

# 대잔치




SKC 제공
TEL : 756 - 5151

균일한 자성체의 배양과 성장으로
도막의 균일도 및 기록신호의
안정도를 극대화하였다.

월간 퍼스널컴퓨터는
독자 여러분의 의견을 소중히 하여
책의 편집에 반영하고자 합니다.
애독자 설문지를 사용하여
질의내용에 성의껏 답하시어
15일까지 보내주시면 27명을 뽑아
선물을 드립니다.

※ 서울 거주자는 6월30일까지 신분증을 지참하시고 직접 본사로 나오시면
해당 상품을 드리며, 지방에 계신 분은 우송해 드리겠습니다.
기간은 6월30일까지이며, 이후는 무효로 처리합니다.

※ 애독자 설문지는 247페이지에 있습니다.

## 애독자 선물대잔치
### 5월호 당첨자 명단

**● 컴퓨터 보안기**

송종훈 경기도 용인군 내사면 남곡리 167 (주)동아전기 기술부
이형영 서울 중구 남대문로 3가 110
정광천 인천시 북구 작전동 뉴서울 아파트 6-111

**● MS-DOS 입문**

박종한 서울시 동작구 신대방2동 361-26 9/4
이용천 인천시 남동구 간석3동 1번지 26/2
전재영 서울 노원구 상계6동 675 대림아파트 1-904
정진우 경기도 평택군 포승면 원정리 평택화력 화학부
허홍석 전남 목포시 죽교2동 72-18 7/1

**● 디스켓 박스(10장들이)**

김준홍 대구시 북구 대현3동 327-32
심용주 서울 도봉구 번1동 457-62
유명준 서울 용산구 원효로3가 256번지
이덕준 경남 울산시 중구 학산동 13-16 9/2
한승종 경기도 안산시 이동 375번지

**● 마우스**

최영철 경기도 과천시 부림동 주공아파트 714-101

**● 디스켓 자켓(4 장들이)**

김병균 경남 진주시 동성동 1-1
남궁욱 서울 영등포구 도림2동 217-5 16/1
류동기 대구시 동구 효목 1동 176-13

**● 컴퓨터 용어사전**

김동열 경남 울주군 농소면 호계리 867-9
김진욱 서울 종로구 신교동 6-35 11/2
안성현 경남 진해시 신흥동 APT 16동 105호
이종곤 대구시 북구 노원3가 227번지(동아공업사내)
이종찬 광주시 동구 남동 132-2 6/2

**● 화신컴퓨터 보안기**

김승욱 서울 마포구 하수동 17-21
이상훈 대구시 남구 봉덕3동 619-2 봉덕아파트 B-203
이창희 서울 종로구 혜화동 고려대학교 의대 본과4년
박진양 강원도 춘천시 후평동 98-12
홍종철 서울 강서구 화곡2동 159-53 21/4

# 책을 마감하고

며칠전 의대에 다니는 친구가 퍼스널 컴퓨터를 구입하는데 도움을 청해와 짤막한 지식이나마 도움이 되고자 판매처를 같이 가고 시스템도 만지작 거리면서 컴퓨터라고는 C자도 모르는 그 친구의 불안감을 덜어주고자 하였다.

그러나 그 친구는 몹시도 불안하고 자신을 믿지 못하는 눈치였다. 물론 누구나 처음 접하는 것에는 선뜻 나서는 것보다는 그 이후의 상황에 대하여 두려움이 앞서기 마련이다. 솔직히 나자신도 그 친구가 과연 이 컴퓨터를 잘 쓸 것인지 의문이었고 괜히 고가의 오락기구 장만하는 것을 부추긴 것은 아닐까? 아니면 똑똑한 친구의 기만 죽이고 볼품없는 고가의 장식품을 사게 하지는 않은 것인가 하고 고민하기도 했었다.

컴퓨터의 보급이 증가하고 있다고는 하지만 많은 사람들이 컴퓨터 앞에서는 주눅들고 있는 현실을 생각해 보면 정보화시대의 이상은 아직 성급한 이야기가 아닌가 의심할 지경이다. 모든 컴퓨터 초보자들에게 권하고 싶은 한마디.

"컴퓨터 앞에서 담대하라. 그러면 컴퓨터를 정복할 수 있으리라"

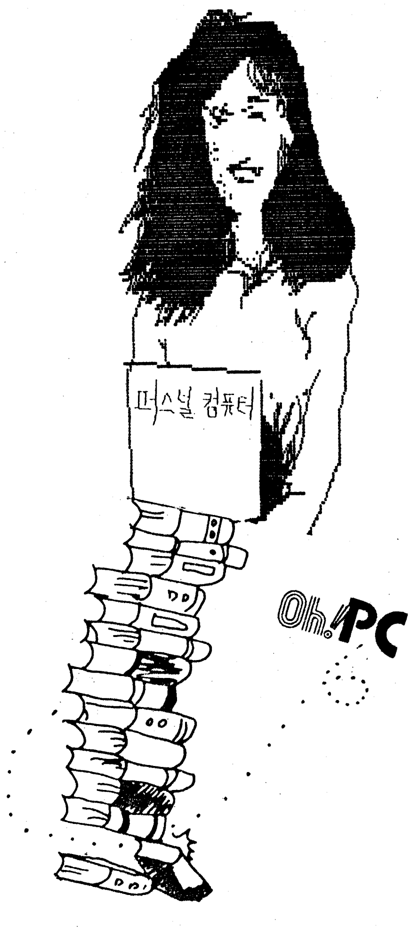
그동안 연재되었던 "둘리가 본 컴퓨터세계"가 끝나게 되어 6월호를 매우 섭섭하게 마감하게 되었다. 그동안 독자들의 인기를 차지했던 연재였는데… 원고를 집필해 주신 이만희 선생님께 감사드리며, 더욱더 알찬 내용으로 독자들이 컴퓨터와 친숙해지기를 둘리가 약속드립니다. 〈둘리〉

일곱!
뜨거운 계절 여름이 온다.
PC도 뜨거워지고, 나또한 뜨거워진다.
몇 년 전 겨울
온 아랫벌에 쌓인 눈을
내 뜨거운 심장으로 녹이려다
혹독한 감기에 걸린적이 있다.
여하튼 그런저런 이유로
난 여름을 좋아한다.
여름은
뜨겁고 화끈한 사나이의 계절, PC와 신경전을 벌이는데 적절한 계절이 될 것이다. 〈榮〉

新綠의 짙푸른 季節에
금새 이리 찬란하게 변한
大自然을 바라보는 것으로
커다란 기쁨이었다.

지난 歲月의 여러 모습들은
저만치 사라지고,
곱게 단장한 자태로
새악시처럼 다소곳
방실방실 웃으며 다가왔다.

어느덧 지난 苦痛은
송두리째 접어두고,
하늘까지 치솟는 氣象은
물고기 지느러미처럼 살포시
가슴에 와닿는다.




깔깔거리는 童心의
가슴속에 파고든 유월은
사방팔방으로 메아리쳐
그 짙푸름을 뽐내고 있다.

아듀~ 〈鎔〉

정말 힘겹게 그리고 아슬아슬하게 그러나 여유있게 보낸 달이었읍니다.

처음부터 꼬리를 물고 일어난 사건들과 외적인 행사(?)들로 我氣는 땅으로 곤두박질쳤고, 변덕스러운 날씨만큼이나 변덕스러운 현의 모습을 보며 많은날 반성, 또 반성했읍니다.

7월호는 조금 긴장해서 일정에 차질이 없도록 노력해야겠습니다. 〈김현〉

내일이 있기에
오늘이 있듯이
오늘, 하루 하루를 충실히 살고싶다.
거짓없이
마음을 비우며
그 무엇에
속박됨이 없이
새싹이 움이트여
나무가 될때 파랗게, 빨갛게
물이 들면
언젠가 이는 바람에
흩날리어 어디론가 이리저리 방황하다가
자취도 없이 사라지리라.

인생도 그러하리
왔다가 가버리는-
하지만, 내일은
또 다시 태양이 떠오리라……. 〈淑〉

점점 긴박해져 오고, 안팎으로 시끄러워, 산다는 것이 얼마나 어렵다는 것을 실감하는 요즘 작은 사랑조차도 스며들 자리가 없다. 버려져 있는 감사의 마음, 잃어버렸던 사랑의 마음을 서로 서로 챙겨보아야 하지 않을까.

이렇게 엉클어진 문제도 한발짝 물러설줄 아는 사랑과 조그마한 일에도 감사해 하는 마음으로 푼다면 산다는 것이 조금 더 쉬워지지 않을까. 〈花〉